UNIVERSITE DE PARIS-VIII

*U.E.R. d'Informatique et de Linguistique*

# VLISP-10

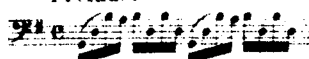
## MANUEL DE RÉFÉRENCE

**RAPPORT TECHNIQUE 17-76**

**JEROME CHAILLOUX**

**MARS 1976**

TABLE DES MATIERES




# 0 - INTRODUCTION

VLISP-10 est une incarnation du modèle de système LISP, nommé VLISP, conçu, mis en place et développé sur plusieurs mini-ordinateurs à l'Université de Paris 8 - Vincennes. C'est dans cette version, l'ordinateur PDP 10 qui en incarne les vertus.

VLISP-10 est un système conversationnel très amical et chaleureux. Sa préoccupation constante est d'aider les gens.

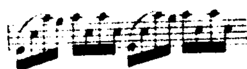
VLISP-10, conscient de sa valeur, n'en est pas moins de taille modeste (3k) permettant ainsi à ses usagers d'utiliser pour leurs besoins la plus grande partie de la mémoire disponible.

VLISP-10 est un interprète très rapide, en conséquence, des applications dont la lenteur serait décourageante sur d'autres systèmes LISP sont exécutées en VLISP dans un temps raisonnable.

VLISP-10 est un système naturel : son implémentation est définie formellement, la démonstration de sa correction n'est pas un horizon inaccessible.

VLISP-10 est un système optimiste et anti-autoritaire. Il suppose que ses utilisateurs sont tout à la fois audacieux et responsables, en conséquence, il donne accès à l'utilisateur aux primitives d'implémentation du système et les protections y sont très réduites : nous n'avons jamais eu à regretter cette décision.

VLISP-10 est un système effectivement utilisé par ses propres implémenteurs pour plusieurs projets d'Intelligence Artificielle en particulier, et de traitements symboliques en général.

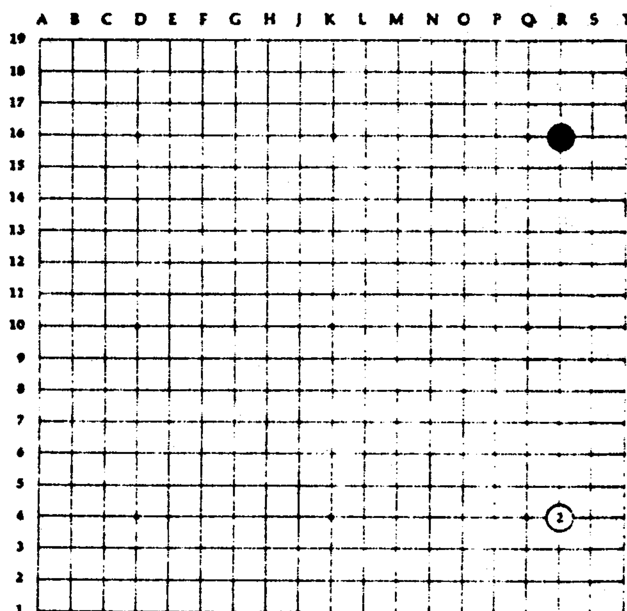


VLISP-10 dispose d'un mode de sortie en braille. Les programmeurs système mal-voyants y sont les bienvenus.

VLISP-10 permet d'écrire des ensembles de fonctions LISP transportables sur tout autre système VLISP.

VLISP-10 serre au plus près l'actualité de la recherche et rend disponibles les développements récents de LISP : contextes, filtrages, démons, acteurs, ...

VLISP-10, enfin, vous souhaite la bienvenue.







# 1 - DESCRIPTION DE L'INTERPRETE

## 1.1 - LES TYPES

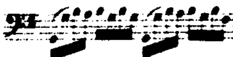
La structure de base LISP est la S-expression (expression symbolique) qui peut être :

- un atome littéral ou numérique
- une paire pointée de S-expressions : ( *s1* . *s2* )
- une liste de S-expressions ( *s1* *s2* ... *sN* )

Les délimiteurs de S-expressions sont l'espace ⊓, le point ., la parenthèse ouvrante ( et la parenthèse fermante ).

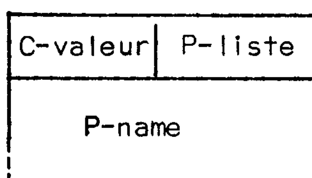
### 1.1.1 - Les atomes littéraux

Ils jouent le rôle des identificateurs de certains autres langages. Ils sont créés implicitement dès leur lecture dans le flux d'entrée ou explicitement par la fonction GENSYM.
Leurs noms (P-name ou Print Name) est une suite de caractères quelconques (contenant au moins un caractère non-numérique). A leur création, seuls les 18 premiers caractères sont pris en compte. On peut insérer dans un P-name des délimiteurs si on les fait précéder du caractère /. Le / peut bien entendu être "slashifié" lui-même.
Chaque atome possède une valeur propre : sa C-valeur qui est indéfinie à sa création, ainsi qu'une liste de propriétés : sa P-liste, qui contient certains aspects particuliers de cet atome, par exemple la ou les fonctions qui lui sont associées.





Un atome littéral a approximativement la structure suivante :




En VLISP-10 le **CAR** d'un atome est sa C-valeur, le **CDR** sa P-liste. Ces atomes sont stockés dans une zone fixe gérée dynamiquement. Certains atomes sont déjà connus de l'interprète :

- les constantes littérales qui contiennent leur propre adresse en C-valeur :

  NIL UNDEF T LAMBDA FSUBR SUBR EXPR FEXPR MACIN MACOUT QUOTE ;

- les fonctions standard (voir 2.)

1.1.2 - Les nombres

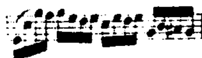
Actuellement VLISP-10 n'utilise que des nombres entiers. Ils sont stockés dans une zone spéciale gérée dynamiquement et sont représentés sur 36 bits (dont 1 bit de signe). Ils doivent être compris dans l'intervalle :

$$\left[- 2^{35} - 1 \quad , \quad + 2^{35}\right]$$

Le P-name d'un nombre est la représentation de sa valeur dans la base de conversion courante (en général 10). La valeur d'un nombre est ce nombre lui-même. Un nombre n'a ni C-valeur ni P-liste.

1.1.3 - Les chaînes

VLISP-10 possède un troisième type d'atomes, les chaînes de caractères. Elles sont décrites en 3..





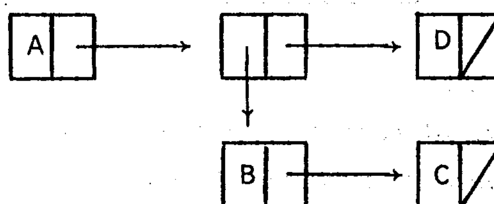
1.1.4 - Les listes

Les listes sont représentées d'une manière standard ; les différents éléments d'une liste utilisent un doublet dont la partie gauche (le CAR) contient l'élément et la partie droite (le CDR) contient un pointeur sur l'élément suivant ou NIL pour le dernier élément.

Exemple :

La liste : est représentée :

(A (B C) D)

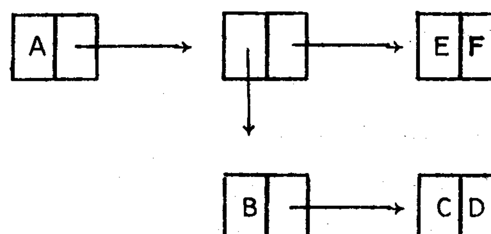


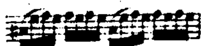

Les paires pointées ( *s1* . *s2* ) permettent de décrire des doublets entiers avec *s1* en partie gauche et *s2* en partie droite.

Exemple :

La liste : est représentée :

(A (B C . D) E . F)







## 1.2 - FONCTIONNEMENT DE L'INTERPRETE

La boucle principale de l'interprète est l'appel de la fonction standard **TOPLEVEL**, indéfiniment. Cette fonction redéfinissable à volonté lit une S-expression, l'évalue en utilisant la fonction interprète **EVAL**, et imprime la valeur de cette évaluation.

La S-expression lue peut être un atome ou une liste.

### 1.2.1 - Evaluation d'un atome

La valeur d'un atome littéral est sa C-valeur. L'évaluation d'un atome littéral dont la C-valeur est indéfinie (auquel on n'a pas encore donné de valeur) provoque à son évaluation une erreur avec impression du message :

** UNDEFINED VARIABLE : l'atome indéfini

La valeur d'un nombre ou d'une chaîne est ce nombre ou cette chaîne elle-même.

### 1.2.2. - Evaluation d'une liste

La fonction **EVAL** considère toujours une liste comme un appel de fonction. Cette liste s'appelle une forme. Le **CAR** de la forme est la fonction, le **CDR** de la forme les arguments de la fonction.
La valeur d'une forme est la valeur ramenée par l'application de la fonction à ses arguments.

1.2.3 - Les Fonctions

La fonction (le CAR de la forme) peut être un atome littéral dans le cas d'une fonction prédéfinie (standard ou utilisateur) ou une liste qui est :

- soit une λ-expression qui est la déclaration explicite d'une fonction
- soit une nouvelle forme dont la valeur est la fonction demandée.

Pour tous ces cas les arguments (le CDR de la forme) peuvent être évalués ou non.

1.2.3.1 - Les fonctions standard

Elles sont ecrites en langage machine et sont résidentes dans l'interprète. Il y a deux types de fonctions standard :

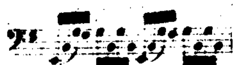
- les { SUBRs qui ont un nombre fixe d'arguments / NSUBRs qui ont un nombre quelconque d'arguments } évalués
- les FSUBRs qui ont un nombre quelconque d'arguments non évalués
  Ces dernières en général ne feront évaluer que certains de leurs arguments ce qui permet à l'interprète d'avoir des structures de contrôle conditionnelles.

Ces fonctions standard sont décrites en 2.

1.2.3.2 - Les fonctions utilisateur

L'utilisateur peut se définir ses propres fonctions en associant à un atome une λ-expression ; plus précisèment en mettant sur la P-liste de cet atome une λ-expression sous l'un des indicateurs suivants :

- EXPR pour une fonction qui évalue ses arguments ;
- FEXPR pour une fonction qui ne les évalue pas ;
- MACIN pour les macro-fonctions d'entrée ;
- MACOUT pour les macro-fonctions de sortie.



1.2.3.3. Les λ-expressions

Une λ-expression est une liste de la forme :

(LAMBDA variables *s1* ... *sN*)

L'exécution d'une λ-expression se fait en 3 étapes :

-1) liaison des arguments (paramètres actuels) évalués ou non, aux variables (paramètres formels). Comme dans tous les VLISPs cette liaison n'utilise pas de A-liste. Avant chaque liaison l'ancienne valeur de la variable est sauvée dans une zone de travail, la liaison peut donc s'effectuer dans la C-valeur. L'accès aux variables devient extrêmement rapide, par contre l'utilisation des **FUNARGs** du LISP 1.5. devient impossible ;

-2) évaluation en séquence des différentes expressions ***s1* ... *sN*.** La valeur de la dernière évaluation ***sN*** sera la valeur retournée par la fonction ;

-3) restitution des anciennes valeurs des variables.

1.2.4 - Liaison des arguments

Les arguments sont liés aux variables d'une manière spécifique à chaque type de fonction.

1.2.4.1 - Liaison des EXPRs

A l'appel d'une fonction de type EXPR, *tous* les arguments sont évalués puis rassemblés en une liste des valeurs. Cette liste est liée au paramètre formel qui peut être :

- une liste de variables. Dans ce cas la liaison s'effectue élément par élément entre la liste des variables et la liste des valeurs. Si la liste des valeurs est plus grande que celle des variables, les valeurs restantes sont ignorées, ceci permet de faire des effets de bord. Si la liste des variables est plus grande que celle des valeurs, les variables





en trop sont liées à **NIL** et font office de variables locales ;

- une variable. Dans ce cas la liste entière des valeurs est liée à cette variable ;
- une combinaison de ces deux types (voir le dernier exemple).

Exemple :

```
EXPR      ? ((LAMBDA (X Y Z)(PRINT X Y Z)) 1 2 3 (PRINT '(EFFET DE BORD)))
            (EFFET DE BORD)
            1 2 3
          = 3

EXPR      ? ((LAMBDA (X Y Z)(PRINT X Y Z)) 1)
            1 NIL NIL
          = NIL

EXPR      ? ((LAMBDA X (PRINT X)) 1 2 3 4)
            (1 2 3 4)
          = (1 2 3 4)

EXPR      ? ((LAMBDA (X Y . Z)(PRINT X Y Z)) 1 2 3 4)
            1 2 (3 4)
          = (3 4)
```

1.2.4.2 - Liaison des FEXPRs

Pour ce type de fonctions aucun argument n'est évalué. Tous les arguments sont rassemblés en une liste qui est liée à la première variable du paramètre formel (qui doit être obligatoirement une liste). Les autres variables de cette liste font office de variables locales et sont initialisées à NIL.





Exemple :

```
FEXPR       ? ((LAMBDA (X)(PRINT X))(CAR '(A B)))
              (CAR '(A B))
            = (CAR '(A B))

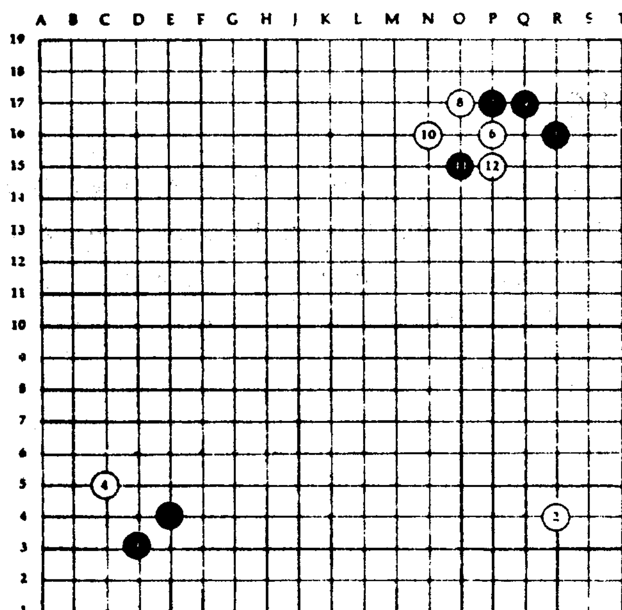
FEXPR       ? ((LAMBDA (X Y . Z)(PRINT X Y Z))(ADD1 2))
              (ADD1 2) NIL NIL
            = NIL
```

1.2.4.3 - Liaison des MACINs MACOUTs

Ces liaisons sont identiques aux EXPRs mais les arguments (le CDR de ce qui a été lu ou devait être imprimé)ne sont pas évalués.

## 1.3 - UTILISATION DE L'INTERPRETE

Actuellement l'interprète VLISP-10 utilisé à Vincennes se sert du projet programmeur 630,300 . Sa version .SAV ainsi que la plupart des utilitaires sont stockés sous ce numéro de pg.pj. Il fonctionne en mode conversationnel à partir du terminal lourd de l'Université.



### 1.3.1 Fonctionnement en mode conversationnel

Il faut au préalable se mettre sous contrôle moniteur en exécutant le LOGIN. Dans le dialogue ci-dessous, le texte frappé par l'utilisateur est souligné.

```
.RUN VLISP [630,300]      {activation de l'interprète sous contrôle moniteur
     VLISP 10
        :                 {identification de l'interprète
        :
  --- ALLO ? ---
?  ()                     {entrée d'une S-expression
=  NIL                    {impression du résultat de son évaluation
?  :
   :
?  (STOP)                 {rappel du moniteur
   EXIT
.                         {le moniteur attend une commande
```

### 1.3.2 - Caractères spéciaux sur la TTY

L'interprète utilise la routine de service standard de gestion des terminaux. L'utilisateur peut donc modifier les caractéristiques du terminal au moyen de la commande moniteur TTY , par exemple la largeur du champ d'impression, le transcodage minuscule-majuscule, ...

En utilisation normale :

- on peut utiliser indifféremment les caractères majusucles ou minuscules ;
- le caractère RUB-OUT permet d'enlever le dernier caractère frappé ;
- le caractère CONTROL-R fait réécrire la ligne en cours (en cas de surabondance de RUB-OUTs par exemple);
- le caractère CONTROL-U permet de recommencer une ligne ;
- les caractères RETURN et ESCAPE signalent la fin de la ligne ;
- les caractères CONTROL-C et ETX rappellent le moniteur. Il vaut mieux utiliser la fonction STOP qui de plus ferme tous les fichiers ouverts.



### 1.3.3 - Utilisation des autres périphériques

L'utilisation directe de l'imprimante (LPS11) et du lecteur de cartes (CDS11) est à déconseiller actuellement car cela perturbe le fonctionnement du moniteur. Si votre programme VLISP est sur cartes et si vous désirez les résultats sur l'imprimante

| à la place de : | faites : | |
|---|---|---|
| | .R PIP<br>*PROG.VLI/T=CDS11:<br>*↑C | création d'un fichier source sur disque à partir des cartes |
| .RUN VLISP [630,300]<br>VLISP 10<br>:<br>--- ALLO ? --- | .RUN VLISP [630,300]<br>VLISP 10<br>:<br>--- ALLO ? --- | activation de l'interprète |
| ? (OUTPUT '(LPS11)) | ? (OUTPUT '(DSK (PROG.LST))) | ouverture du fichier résultat sur disque |
| :<br>? (INPUT '(CDS11))<br>*** E.O.F.<br>--- ALLO ? --- | :<br>? (INPUT '(DSK (PROG.VLI)))<br>*** E.O.F.<br>--- ALLO ? --- | exécution du fichier source sur disque |
| ?<br>: | : | passage en mode conversationnel |
| ? (STOP)<br>EXIT | ? (STOP)<br>EXIT | rappel du moniteur |
| . | .R PIP<br>*LPS11:=PROG.LST<br>*↑C<br>. | édition sur imprimante du fichier résultat sur disque |

### 1.3.4 - Initialisation de VLISP

Avant de commencer le dialogue sur la TTY, VLISP-10 va lire sur disque le fichier VLISP.INI, s'il existe, sous le numéro de projet-programmeur de l'utilisateur.

Ce fichier peut contenir les initialisations souhaitées par l'utilisateur, une simulation de Batch, etc.



## 1.4 - LES ERREURS

Les erreurs sont détectées par le moniteur ou par l'interprète.

### 1.4.1 - Erreurs détectées par le moniteur

Pour toutes ces erreurs un message est imprimé sur la TTY puis le contrôle est rendu au moniteur qui se met en attente d'une commande. Si l'erreur n'est pas fatale, on peut relancer l'interprète avec la commande .REE sinon il faut recharger un interprète neuf avec la commande .RUN VLISP.

? PC OUT OF BOUNDS AT USER PC xxxx
? NON EX MEM AT USER PC xxxx
? MEM PAR ERROR AT USER PC xxxx

> Si l'une de ces trois erreurs apparaît, il faut absolument recharger l'interprète, le système est dans un très mauvais état.

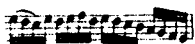
? ILL REF MEMORY AT USER PC xxxx

> Cette erreur apparaît si l'utilisateur exécute des recherches sur des objets indéterminés ou sur des structures altérées.
> ex : (CDDR 99999)

? PDL OV AT USER PC xxxx

> La pile de travail de l'interprète est saturée. Cette erreur est due en général à une récursion infinie et n'est pas fatale.
> ex : le fameux combinateur :

```
? ((LAMBDA (X)(X X)) '(LAMBDA (X)(X X)))
  ? PDL OV AT USER PC xxxx
  .REE
?
```



## 1.4.2 - Erreurs détectées par l'interprète

Pour toutes ces erreurs un message est imprimé sur le fichier de sortie courant, l'évaluation qui a déclenché cette erreur est stoppée et le contrôle est rendu à la boucle principale de l'interprète qui, rappelons-le, appelle la fonction TOPLEVEL.

### 1.4.2.1 - Erreurs d'entrée/sortie

*** READ ERROR.

erreur de lecture. La S-expression lue est syntaxiquement incorrecte. En général cette erreur est due à une mauvaise écriture des paires pointées.

*** OPEN ERROR (INPUT).

erreur à l'ouverture du device d'entrée.

*** OPEN ERROR (OUTPUT).

erreur à l'ouverture du device de sortie.

*** LOOKUP ERROR (INPUT) : *n*

erreur à l'ouverture du fichier d'entrée.

*** ENTER ERROR (OUTPUT) : *n*

erreur à l'ouverture du fichier de sortie.

### 1.4.2.2 - Erreurs à l'évaluation

*** UNDEFINED VARIABLE : *a*

l'atome *a* est indéfinie. Il n'a pas de valeur.

*** UNDEFINED FUNCTION (EVAL) : *a*

l'atome *a* ne possède pas de définition de fonction. Cette erreur est déclenchée par la fonction interprète **EVAL**.

*** UNDEFINED FUNCTION (APPLY) : *a*

l'atome *a* ne possède pas de définition de fonction. Cette erreur est déclenchée par la fonction interprète **APPLY**.

*** UNDEFINED FUNCTION (SUBR) : *a*

l'atome *a* ne possède pas de définition de fonction de type SUBR. Cette erreur est déclenchée par la fonction **SUBR**.

*** UNDEFINED FUNCTION (FSUBR) : *a*

l'atome *a* ne possède pas de définition de fonction de type FSUBR. Cette erreur est déclenchée par la fonction **FSUBR**.

*** STATUS ERROR : *n*

on a donné à la fonction **STATUS** un mauvais argument.

*** LABEL ERROR : *a*

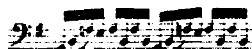
l'atome *a* n'est pas une etiquette connue ou bien on a utilisé un GO ou un **GOTO** à l'extérieur d'un **PROG**.

*** RETURN ERROR.

l'utilisateur tente de faire un **RETURN** à l'extérieur d'un **PROG**.

*** ESCAPE ERROR.

l'utilisateur tente de faire un **LESCAPE** ou d'utiliser une fonction d'échappement alors qu'il n'y a plus de λ-expression active.



### 1.4.2.3 - Erreur dans la gestion de la mémoire

**:: ATOM ERROR.**

il n'y a plus de place pour stocker les atomes.

**:: FULL MEMORY.**

il n'y a plus de place pour stocker les listes.

**:: USER STACK OVERFLOW.**

débordement de la pile utilisateur.

**:: USER STACK UNDERFLOW.**

la pile utilisateur est vide.

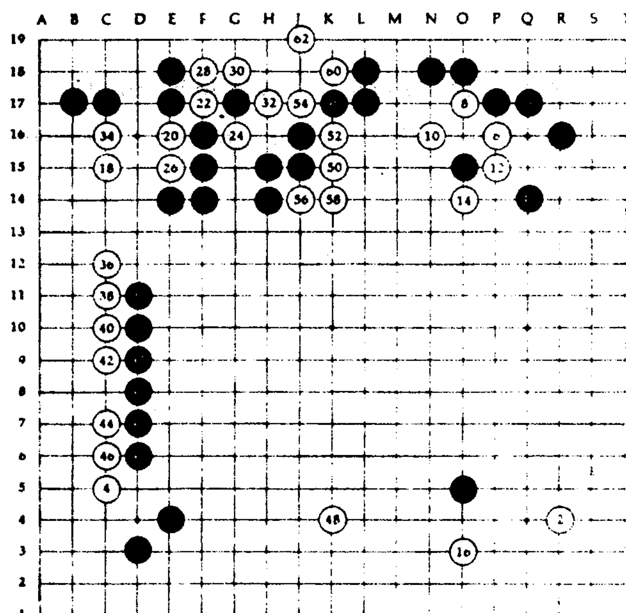
### 1.4.2.4 - Erreurs contrôlées par l'utilisateur

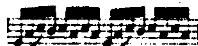
**:: G.C. STEP DONE**

le nombre limite de garbages collecting autorisé par l'utilisateur est atteint.

**:: USER ERROR :** *s*

l'utilisateur a appellé la fonction ERROR.





## 2 - DESCRIPTION DES FONCTIONS STANDARD

Toutes les fonctions qui vont être décrites dans ce chapitre sont résidentes dans le système actuel.

Pour chacune d'elles, on donnera :

- le type de la fonction
  - SUBR
  - NSUBR
  - FSUBR
- le type souhaité des arguments :
  - *s* une S-expression quelconque
  - *l* une liste
  - *a* un atome (atome littéral, nombre ou chaîne)
  - *c* un caractère (atome littéral dont le P-name est mono-caractère)
  - *fn* une fonction

Le système ne contrôle pas le type et la valeur des arguments fournis à l'appel de ces fonctions (sauf pour les fonctions status).



## 2.1 - LES FONCTIONS INTERPRETES

(EVAL *s*) - SUBR -

C'est la fonction principale de l'interprète. EVAL ramène la valeur de l'évaluation de *s* (qui est évalué).

ex : ? (EVAL '(CONS 'A 'B)) → (A . B)

(APPLY *fn l*) - SUBR -

ramène la valeur de l'application de la fonction *fn* à la liste d'arguments *l* qui sont évalués. La fonction *fn* doit être une SUBR, une EXPR ou une λ-expression.

ex : ? (APPLY 'CONS '(A B)) → (A . B)

(EVLIS *l*) - SUBR -

ramène la liste composée des valeurs des évaluations de tous les éléments de la liste *l*.

```
ex : ? (SETQQ L ((PRIN1 1)(PRIN1 2)(PRIN1 3)))
     → ((PRIN1 1)(PRIN1 2)(PRIN1 3))
     ? (EVLIS L) 1 2 3 → (1 2 3)
```

(EPROGN *l*) - SUBR -

évalue tous les éléments de la liste *l*. Ramène la valeur de la dernière évaluation.

```
ex : ? (SETQQ L ((PRIN1 1)(PRIN1 2)(PRIN1 3)))
     → ((PRIN1 1)(PRIN1 2)(PRIN1 3))
     ? (EPROGN L) 1 2 3 → 3
```

(PROGN *s1* ... *sN*) - FSUBR -

évalue les différentes expressions *s1* ... *sN*. Ramène la valeur de la dernière évaluation, c'est-à-dire celle de *sN*.

```
ex : ? (PROGN (PRIN1 1)(PRIN1 2)(PRIN1 3)) 1 2 3
     → 3
```





(QUOTE *s*) - FSUBR -

ramène *s* non-évalué. Il existe un macro-caractère qui facilite cette écriture : '*s*. (voir 2.15.7)

ex : ? (QUOTE A) → A

(ID *s*) - SUBR -

c'est la fonction identique.ID ramène *s* qui est évalué.

ex : (ID 'A) → A

(SUBR *fn* *l*) - SUBR -

applique la définition de type SUBR de la fonction *fn* à la liste d'arguments *l*. Si *fn* n'a pas de définition de type SUBR, une erreur apparaît avec impression du libellé :

*** UNDEFINED FUNCTION (SUBR) : *fn*

ex : ? (SUBR 'ADD1 '(3)) → 4

(FSUBR *fn* *l*) - SUBR -

identique à SUBR mais *fn* doit avoir une définition de type FSUBR. Si ce n'est pas le cas le libellé d'erreur est

*** UNDEFINED FUNCTION (FSUBR) : *fn*

Ces deux fonctions sont utilisées pour lancer la version standard des fonctions standards redéfinies par l'utilisateur.

ex : ? (SETQQ N 3) → 3

? (FSUBR 'INCR '(N)) → 4



## 2.2 Les prédicats de base

### 2.2.1 Test sur les types

Tous sont de type SUBR et ramènent T (vrai) ou NIL (faux) en fonction du résultat du test.

| | vrai si |
|---|---|
| (NULL *s*) | *s* = NIL |
| (NOT *s*) | *s* = NIL (identique à NOT) |
| (ATOM *s*) | *s* est un atome (atome littéral, nombre ou chaîne) |
| (LITATOM *s*) | *s* est un atome littéral |
| (LISTP *s*) | *s* est une liste<br>(LISTP *s*) est équivalent à (NOT (ATOM *s*)) |

### 2.2.2 Comparaisons

(EQP *s1 s2*) - SUBR -

sert à tester 2 atomes litteraux. EQP ramène T si *s1* et *s2* sont les mêmes atomes littéraux, sinon ramène NIL. Si *s1* et *s2* sont des nombres, des chaînes ou des listes, EQP teste si *s1* et *s2* ont la même adresse physique.

ex : (EQP 'A (CAR '(A))) → T
(EQP '(A B) '(A B)) → NIL

(NEQP *s1 s2*) - SUBR -

est équivalent à (NOT (EQP *s1 s2*)).





(EQ *s1 s2*) - SUBR -

sert à tester 2 atomes (de n'importe quel type). EQ ramène T si les 2 atomes *s1* et *s2* sont égaux, sinon ramène NIL.

2 atomes littéraux sont égaux s'ils ont le même nom.

2 nombres sont égaux s'ils ont la même valeur.

2 chaînes sont égales si elles ont la même longueur et les mêmes caractères.

Si *s1* et *s2* sont des listes, EQ teste si *s1* et *s2* ont la même adresse physique (comme pour la fonction EQP).

ex : (EQ 'A (CAR '(A))) → T
(EQ (ADD1 119) 120) → T
(EQ "STRG" "STRG") → T
(EQ '(A B) '(A B)) → NIL

(NEQ *s1 s2*) - SUBR -

est équivalent à (NOT (EQ *s1 s2*))

(EQUAL *s1 s2*) - SUBR -

sert à tester 2 listes. EQUAL ramène T si les 2 S-expressions *s1 s2* ont la même structure, sinon ramène NIL.

ex : (EQUAL '(A B (C . D)) '(A B (C . D)) → T

(NEQUAL *s1 s2*) - SUBR -

est équivalent à (EQ *s1 s2*))

(SORT *a1 a2*) - SUBR -

ramène T si le P-name de l'atome littéral *a1* est plus petit ou égal lexicographiquement que le P-name de l'atome littéral *a2*. Cette fonction est utilisée pour faire des tris alphabétiques.

ex : ? (SORT 'AA 'AB) → T
? (SORT 'AAA 'AAA) → T
? (SORT 'ABC 'AZ) → NIL





## 2.3 LES FONCTIONS DE CONTROLE

### 2.3.1 Les fonctions de contrôle de base

(OR *s1* ... *sN*) - FSUBR -

évalue successivement les différentes expressions *s1* ... *sN* jusqu'à ce que l'une de ces évaluations ait une valeur différente de NIL. OR ramène cette valeur. OR permet de construire une structure de contrôle de type

si non *s1* alors si non *s2* alors ... *sN*

ex : ? (OR NIL NIL 2 3) → 2

(AND *s1* ... *sN*) - FSUBR -

évalue successivement les différentes expressions *s1* ... *sN* Si la valeur d'une de ces évaluations est égale à NIL, AND ramène NIL sinon AND ramène la dernière évaluation *sN*. AND permet de construire une structure de contrôle de type

si *s1* alors si *s2* alors ... *sN*.

ex : ? (AND 1 2 3 4) → 4

? (AND 1 2 NIL 3) → NIL

(IF *s1* *s2* *s3* ... *sN*) - FSUBR -

si la valeur de l'évaluation de *s1* est différente de NIL, IF ramène la valeur de l'évaluation de l'expression *s2*, sinon IF évalue les différentes expressions *s3* ... *sN* et ramène la valeur de la dernière évaluation *sN*. IF permet de construire une structure de contrôle de type

si *s1* alors *s2* sinon *s3* ... *sN*.

ex : ? (IF T 1 2 3) → 1

? (IF NIL 1 2 3) → 3





(COND *l1* ... *lN*) - FSUBR -

est l'instruction conditionnelle la plus générale. Les différents arguments *l1* ... *lN* sont des listes, appelées clauses, qui ont la structure suivante

(*c s1* ... *sN*)

COND va sélecter une seule de ces clauses : celle dont l'évaluation de son 1er élément *c* est différente de NIL (est *vraie*).COND évalue alors les différentes expressions *s1* ... *sN* et ramène la valeur de la dernière évaluation *sN*.
Si la clause sélectée n'a qu'un élément *c*, COND ramène la valeur de l'évaluation de *c* (c'est-à-dire la valeur qui a déclenché la sélection de cette clause).COND permet de construire des structures de contrôle de type

si ... alors ... sinon si ... alors ...

Si aucune clause n'est sélectée,COND ramène NIL.

ex : ? (COND (NIL 1 2)
(T 3 4 5)) 5

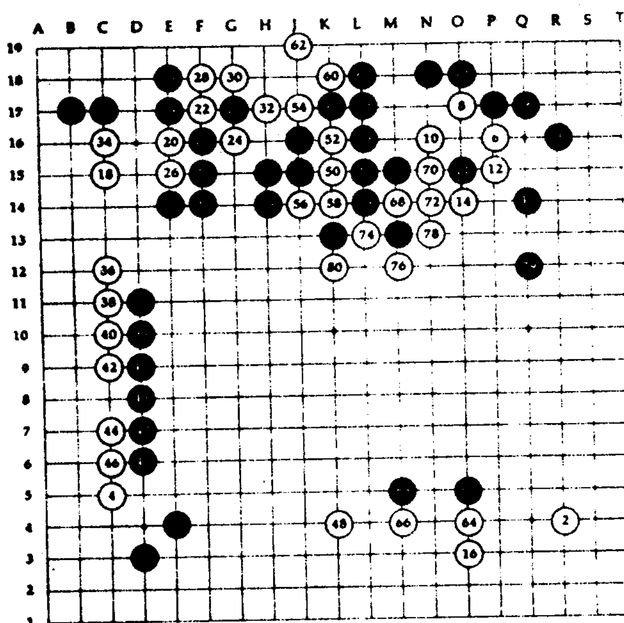
```
(COND
    ((ZEROP X) 'ZERO)
    ((ODDP X) 'IMPAIR)
    ((EVENP X) 'PAIR)
    ('NON))
```

**(SELECT** *s l1 ... lN lF*) - FSUBR -

comme pour la fonction COND, SELECT va sélecter une des clauses *l1 ... lN*. Le sélecteur de ces clauses est la valeur de l'évaluation de *s*, la sélection s'effectue par comparaison (au moyen du prédicat EQUAL ) du sélecteur avec la valeur de l'évaluation du 1er élément de chacune des clauses *l1 ... lN*. Dès qu'une clause est sélectée, SELECT évalue le reste de la clause et ramène la valeur de la dernière évaluation, i.e. (EPROGN (CDR *clause*))

Si aucune des clauses *l1 ... lN* n'est sélectée, la dernière clause *lF* est évaluée ; SELECT ramène la valeur de (EPROGN *lF*). SELECT permet donc de construire des aiguillages.

ex : 
```
? (SELECT 10
        (12 'BOF)
        ((ADD1 9) 'OUI)
        ('NON))
  → OUI
? (SELECT 11
        (12 'BOF)
        ((ADD1 9) 'OUI)
        ('NON))
  → NON
```





(SELECTQ *s l1* ... *lN lF*) - FSUBR -

comme pour la fonction COND, SELECTQ va sélecter une des clauses *l1* ... *lN*. Le sélecteur de ces clauses est la valeur de l'évaluation de *s*, la sélection s'effectue par comparaison du sélecteur avec :

- le CAR (non évalué) de la clause si ce CAR est atomique (en utilisant le prédicat EQ) ;
- les différents atomes du CAR (non évalué) de cette clause (en utilisant la fonction MEMQ).

Dès qu'une clause est sélectée, SELECTQ évalue le reste de la clause et ramène la valeur de la dernière évaluation (i.e. (EPROGN (CDR *clause*))).

Si aucune des clauses *l1* ... *lN* n'est sélectée, SELECTQ évalue automatiquement la dernière clause *lF* et ramène la valeur du (EPROGN *lF*).

SELECTQ permet de faire des aiguillages sur valeurs constantes.

ex :
```
? (SELECTQ 'ROUGE
        (VERT 'ESPOIR)
        ((BLEU ROUGE JAUNE) 'OK)
        ('NON))
→ OK
? (SELECTQ 'BLANC
        (VERT 'ESPOIR)
        ((BLEU ROUGE JAUNE) 'OK)
        ('NON))
→ NON
```



(WHILE *s s1 ... sN*) - FSUBR -

tant que la valeur de l'évaluation de *s* est différente de NIL, WHILE va évaluer les différentes expressions *s1 ... sN*. WHILE ramène toujours NIL (la dernière évaluation de *s*). Il permet de construire des boucles conditionnelles d'une manière fort commode.

ex : ? (SETQQ L (A B C D)) → (A B C D)
? (WHILE L (PRIN1 (NEXTL L))) A B C D → NIL

(UNTIL *s s1 ... sN*) - FSUBR -

tant que la valeur de l'évaluation de *s* est égale à NIL, UNTIL va évaluer les différentes expressions *s1 ... sN*. UNTIL ramène la valeur de l'évaluation de *s* qui a fait arrêter la boucle. A la valeur ramenée près, UNTIL est équivalent à

(WHILE (NOT *s*) *s1 ... sN*)

ex : ? (SETQQ L (A B 2 C D)) → (A B 2 C D)
? (UNTIL (NUMBP (CAR L)) (PRIN1 (NEXTL L))) A B → 2

(REPEAT *s s1 ... sN*) - FSUBR -

évalue *s*. Cette valeur doit être un nombre *n*. REPEAT évalue alors *n* fois les différentes expressions *s1 ... sN*. Si l'évaluation de *n* n'est pas un nombre strictement positif, la boucle n'est pas exécutée et REPEAT ramène NIL, sinon REPEAT ramène la dernière évaluation de *sN*.

ex : ? (REPEAT 10 (PRIN1 '*)) * * * * * * * * * * → *
? (REPEAT 0 (PRIN1 '*)) → NIL



## 2.3.2 Les fonctions d'échappement

(ESCAPE *a s1* ... *sN*) - FSUBR -

est la fonction de contrôle la plus puissante des systèmes VLISPs. *a* est un atome littéral qui devient le nom d'une fonction d'échappement, puis les différentes expressions sont évaluées en séquence. Si au cours de ces évaluations, une forme de type (*a se1* ... *seN*) est rencontrée, les différentes expressions *se1* ... *seN* sont évaluées et ESCAPE ramène la valeur de la dernière évaluation (i.e. celle de *seN*). Si une telle forme n'est pas rencontrée, ESCAPE ramène la valeur de l'évaluation de *sN*.

ex : ? (ESCAPE EXIT

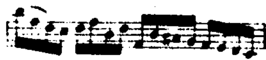
```
        (MAPC '(A B 2 C)
              '(LAMBDA (X)
                  (AND (NUMBP X) (EXIT 'OUI))))
        'NON))
   → OUI
```

(LESCAPE *s1* ... *sN*) - FSUBR -

évalue les différentes expressions *s1* ... *sN* en séquence et ramène la valeur de l'évaluation de *sN*. De plus LESCAPE fait sortir de la dernière λ-expression connue (fonction utilisateur ou λ-expression explicite), avec pour valeur la valeur du LESCAPE.

ex : ? (MAPC '(A B 2 3 C)

```
          '(LAMBDA (X)
              (AND (NUMBP X) (LESCAPE 'OUI))))
   → OUI
```



### 2.3.3 Les fonctions de contrôle de type PROG

Ce type de structure de contrôle permet à l'usager conservateur d'écrire des séquences de LISP sans structure, vertu nocturne partagée avec certains autres langages.
On peut se brancher à des étiquettes (fonction GO et GOTO) et sortir d'un corps de **PROG** comme en FORTRAN (la fonction éternelle :**RETURN** ).
Ce type de structure de contrôle a été conservé dans un souci de compatibilité avec les autres systèmes LISPs. Les fonctions d'échappement des systèmes VLISPs étant à la fois plus puissantes et plus rapides à l'interprétation.

(PROG *l s1 ... sN*) - FSUBR -

*l* est une liste d'atomes qui font office de variables locales à l'intérieur du **PROG**. Elles sont liées à la valeur **NIL** à l'entrée du **PROG** et restaurées à leur ancienne valeur au retour du **PROG**. Cette protection des variables locales ne porte que sur leurs C-val. Cette liste de variables locales peut être vide (égale à NIL) mais ne peut pas être omise.
*s1 ...sN* est une liste de formes qui sont évaluées en séquence. Si dans cette liste se trouvent des atomes, ils sont considérés comme des étiquettes et ne sont donc pas évalués.
La valeur d'un **PROG**, s'il n'est pas interrompu par un **RETURN**, est la valeur de l'évaluation de *sN*.



(GO *a*) - FSUBR -

*a* doit être le nom d'une étiquette existante du PROG. L'évaluation de la forme (GO *a*) fait reprendre l'évaluation à la forme qui suit l'étiquette *a*. Si *a* n'est pas une étiquette connue ou si la forme (GO *a*) apparaît à l'extérieur d'un PROG, le message d'erreur suivant apparaît :

** LABEL ERROR : *a*

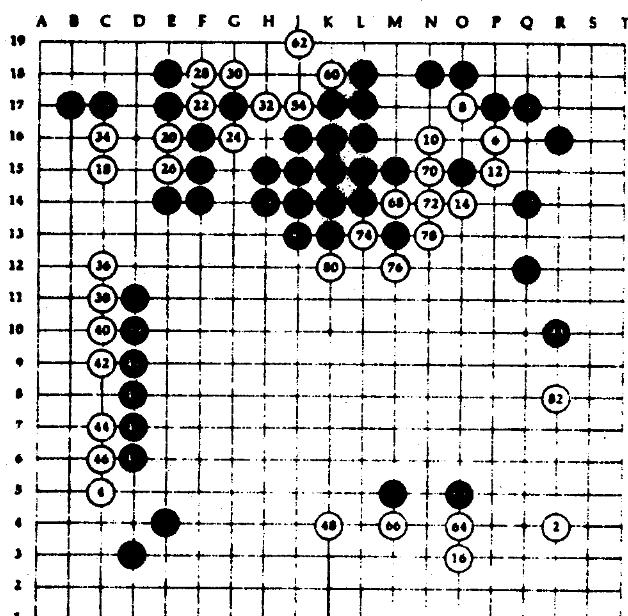
(GOTO *s*) - SUBR -

identique à GO mais l'argument *s* est évalué et doit ramener un atome en valeur.

(RETURN *s*) - SUBR -

sort du PROG courant. La valeur du PROG est celle de l'évaluation de *s*. Si une forme (RETURN *s*) apparaît en dehors d'un PROG, le message d'erreur suivant apparaît :

** RETURN ERROR.





## 2.4. LES FONCTIONNELLES

Toutes sont de type SUBR et utilisent des fonctions *fn* en argument. Ces fonctions peuvent être des fonctions standard SUBR, utilisateurs EXPR ou des λ-expressions à 1 argument.

(SOME *l* *fn*) - SUBR -

applique successsivement la fonction *fn* à tous les éléments de la liste *l* jusqu'à ce qu'une de ces applications donne une valeur différente de NIL qui est ramenée en valeur.

ex : ? (SOME '(A B 2 C) 'NUMBP) → 2

? (SOME '(A B 2 C) 'LISTP) → NIL

(EVERY *l* *fn*) - SUBR -

applique successivement la fonction *fn* à tous les éléments de la liste *l*. Si l'une de ces applications donne une valeur NIL, EVERY ramène NIL sinon EVERY ramène la valeur de la dernière application.

ex : ? (EVERY '(A B 3 C) 'ATOM) → C

? (EVERY '(A B 3 C) 'LITATOM) → NIL

(ORF *l* *fn1* ... *fnN*) - SUBR -

applique les différentes fonctions *fn1* ... *fnN* à l'argument *l* jusqu'à ce qu'une de ces applications donne une valeur différente de NIL qui est ramenée en valeur, sinon ORF ramène NIL.

ex : ? (ORF 3 'LITATOM 'NUMBP 'LISTP) → 3

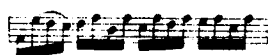
? (ORF 3 'LITATOM 'LISTP) → NIL

(ANDF *l* *fn1* ... *fnN*) - SUBR -

applique les différentes fonctions *fn1* ... *fnN* à l'argument *l*. Si l'une de ces applications donne une valeur égale à NIL , ANDF ramène NIL sinon ANDF ramène la valeur de la dernière application, c'est à dire celle de (*fnN* *l*).

ex : ? (ANDF 3 'GZP 'ODDP) → 3

? (ANDF 4 'GZP 'ODDP) → NIL



Les fonctions d'application.

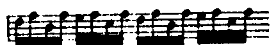
Syntaxe : (MAP ... *l* *fn*)

Elles permettent d'appliquer la fonction *fn* sur certains sous-ensembles de la liste *l*.

| applique la fonction *fn* sur : | | | et ramène en valeur : |
|---|---|---|---|
| *l* puis sur ses CDRs successifs | les CARs successifs de *l* | *l* puis sur toutes ses sous-structures | |
| MAP | MAPC | MAPS | NIL |
| MAPLIST | MAPCAR | MAPSUB | la liste des valeurs de toutes les applications |
| MAPT | MAPCT | MAPST | la liste des valeurs différentes de NIL de toutes les applications |

Les fonctions de type MAPS (elles n'existent qu'en VLISP) permettent d'explorer des structures en arbre. Elles opèrent en DEEP-FIRST sur les CARs.

ex : ? (MAPC '(A (B C) D) 'PRIN1) A (B C) D → NIL
? (MAPST '(A (B 2 (3)) 4 C) 'NUMBP) → (2 3 4)



## 2.5 - LES FONCTIONS DE RECHERCHE

### 2.5.1 - Recherches sur listes

(CAR *s*) - SUBR -

si *s* est un atome ramène sa C-valeur
si *s* est une liste ramène son premier élément
le CAR d'un nombre ou d'une chaîne est indéterminé.
ex : ? (CAR '(A B C)) → A

(CDR *s*) - SUBR -

si *s* est un atome ramène sa P-liste
si *s* est une liste ramène cette liste sans son premier élément
le CDR d'un nombre ou d'une chaîne est indéterminé.
ex : ? (CDR '(A B C)) → (B C)

(C{AAA / DDD}R *s*) - SUBR -

les 12 combinaisons de CAR ou de CDR imbriqués sont disponibles.
(CADR *s*) est équivalent à (CAR (CDR *s*))
(CDDAR *s*) est équivalent à (CDR (CDR (CAR *s*)))

(MEMQ *a* *l*) - SUBR -

ramène NIL si l'atome *a* n'est pas un élément de la liste *l* sinon ramène la partie de *l* commençant à l'atome *a*. Cette fonction utilise le prédicat EQ.
ex : ? (MEMQ 'B '(A B C D)) → (B C D)

(MEMBER *s* *l*)   - SUBR -

ramène NIL si l'expression *s* n'est pas un élément de la liste *l* sinon ramène la partie de *l* commençant à l'expression *s*. Cette fonction est identique à la fonction MEMQ mais utilise le prédicat EQUAL.

ex : ? (MEMBER '(B C) '((A (B C))(B C) D) → ((B C) D)

(NTH *n* *l*)   - SUBR -

ramène la partie de *l* commençant à son *n*ième élément. Si *n*<1, NTH ramène la liste *l* en entier.

ex : ? NTH 3 '(A B C D)) → (C D)

(CNTH *n* *l*)   - SUBR -

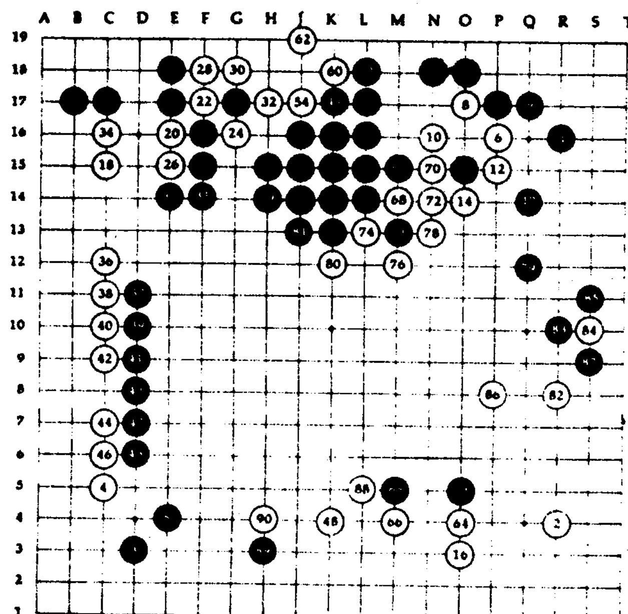
ramène le *n*ième élément de la liste *l*. CNTH est équivalent à (CAR (NTH *n* *l*))

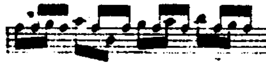
ex : ? (CNTH 3 '(A B C D)) → C

(LAST *l*)   - SUBR -

ramène la liste ne contenant que le dernier élément de la liste *l*.

ex : ? (LAST '(A B C)) → (C)
     ? (LAST '(A B C . D)) → (C . D)





## 2.5.2 - Recherche de type

(TYPEP *s*) - SUBR -

si *s* est un atome littéral ramène l'atome LITATOM

si *s* est un nombre ramène l'atome NUMBP

si *s* est une chaîne ramène l'atome STRINGP

si *s* est une liste ramène l'atome LISTP

ex : ? (TYPEP 'A) → LITATOM

? (TYPEP 3) → NUMBP

? (TYPEP '(A B)) → LISTP

(TYPEFN *a*) - SUBR -

si l'atome *a* possède une définition

d'EXPR ramène l'atome EXPR

de FEXPR ramène l'atome FEXPR

de SUBR ramène l'atome SUBR

de FSUBR ramène l'atome FSUBR

de MACIN ramène l'atome MACIN

de MACOUT ramène l'atome MACOUT

sinon ramène NIL

ex : ? (TYPEFN 'TYPEFN) → SUBR

? (TYPEFN 'COND) → FSUBR

? (TYPEFN 'QUOI) → NIL



## 2.6 - FONCTIONS DE CREATION DE LISTES

(CONS *s1 s2*) - SUBR -

construit une liste dont le premier élément est *s1* et le reste de la liste *s2*. Si *s2* est un atome CONS crée la paire pointée (*s1* . *s2*).

ex : ? (CONS 'A '(B C)) → (A B C)

? (CONS 'X 'Y) → (X . Y)

? (CONS 'Z) → (Z)

(LIST *s1* ... *sN*) - NSUBR -

ramène la liste des valeurs des différentes expressions *s1* ... *sN*.

ex : ? (LIST 'A 'B '(C D) (ADD1 3)) → (A B (C D) 4)

(LINEAR *s1* ... *sN*) -NSUBR -

ramène la liste de tous les atomes des différentes expressions *s1* ... *sN*.

ex : ? (LINEAR 'A '(B (C . D)) '((3))) → (A B C D 3)

(SUBST *s1 s2 l*) - SUBR -

ramène une copie de la liste *l* en substituant à l'expression *s2* l'expression *s1* à chacune de ses occurences. Si *s1* et *s2* sont nulles, (SUBST NIL NIL *l*) ramène une copie de toute la liste *l*.

ex : ? (SUBST '(X Y Z) '(A B) ' ((A B) C (D (A B))))

→ ((X Y Z) C (D (X Y Z)))

(OBLIST *a*) - SUBR -

ramène la (longue) liste de tous les atomes littéraux présents du système à partir de l'atome *a*. Le premier atome système est NIL, le dernier STOP.

(REVERSE *l*) - SUBR -

ramène une copie inversée du top-level de *l*.

ex : ? (REVERSE '(A (B C) D E)) → (E D (B C) A)

(APPEND *l1* *l2*) - SUBR -

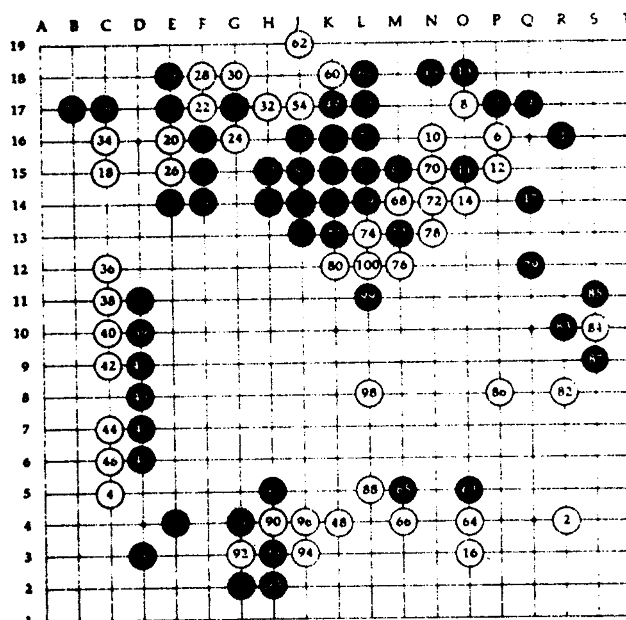
ramène la concaténation d'une copie du top-level de *l1* à *l2*. Si *l2* n'est pas donné, (APPEND *l1*) ramène une copie du top-level de *l1*.

ex : ? (APPEND '(A B) '(C D)) → (A B C D)

(APPEND1 *l1* *s1* ... *sN*) - NSUBR -

ramène la concaténation d'une copie du top-level de *l1* à la liste (*s1* ... *sN*). APPEND1 est équivalent à (APPEND *l1* (LIST *s1* ... *sN*))

ex : ? (APPEND1 '(A B) 'C 'D) → (A B C D)





## 2.7 LES FONCTIONS DE MODIFICATION

Ces fonctions doivent être utilisées conformément au mode d'emploi, pour éviter de placer le système dans un état de confusion dramatique (car elles modifient physiquement les structures VLISPs).

Il est interdit de modifier :

- les constantes symboliques (NIL T LAMBDA ...) et en général il est recommandé de ne pas modifier les atomes systèmes ;
- les nombres ;
- les chaînes de caractères.

Pour ces fonctions l'argument *obj* représente un atome littéral ou une liste. Modifier le CAR d'un atome revient à changer sa C-valeur, modifier son CDR, sa P-liste.

(RPLACA *obj s*) - SUBR -

remplace le CAR de *obj* par *s*. RPLACA ramène *obj* en valeur.

(RPLACD *obj s*) - SUBR -

remplace le CDR de *obj* par *s*. RPLACD ramène *obj* en valeur.

(SET *obj1 s1* ... *objN sN*) - NSUBR -

remplace tous les CARs des *objs* par leur *s* correspondants. SET ramène *sN* en valeur.

(SETQ *obj1 s1* ... *objN sN*) - FSUBR -

identique à SET mais SETQ n'évalue par les différents *objs*.



(SETQQ *obj1 s1 ... objN sN*) - FSUBR -

identique à SET mais SETQQ n'évalue ni les différents *objs*, ni les différents *s*.

(SYNONYM *a1 a2*) - SUBR -

donne à l'atome littéral *a1* les mêmes indicateurs spéciaux (SUBR FSUBR ...) que l'atome *a2*. SYNONYM permet de changer le nom d'une fonction standard,SYNONYM ramène *a1* en valeur.

ex : ? (SYNONYM 'KONSS 'CONS) → KONSS
? (KONSS 1 2) → (1 . 2)
? (CONS 1 2) → (1 . 2)

(NEXTL *a*) - FSUBR -

*a* (qui n'est pas évalué) est un atome littéral dont la valeur (C-val) doit être une liste. NEXTL ramène le CAR de cette liste en valeur et place dans *a* le CDR de son ancienne valeur.

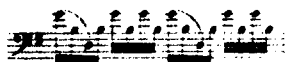
ex : ? (SETQ A '(X Y Z)) → (X Y Z)
? (NEXTL A) → X
? A → (Y Z)

(NEWL *a s*) - FSUBR -

NEWL place en tête de cette liste la valeur de l'évaluation de l'expression *s* et ramène en valeur la liste ainsi formée, i.e. la nouvelle valeur de *a*. Cette fonction est équivalente à (SETQ *a* (CONS *s a*))

L'utilisation conjuguée des fonctions NEXTL et NEWL permet de construire des piles-listes.

ex : ? (SETQ A '(X Y Z)) → (X Y Z)
? (NEWL A 'W) → (W X Y Z)
? A → (W X Y Z)



(INCR *a*) - FSUBR -

l'atome littéral *a* (qui n'est pas évalué) doit avoir un nombre comme valeur. INCR incrémente de 1 la valeur de *a* et la ramène en valeur.

ex : ? (SETQ X 2) → 2
? (INCR X) → 3

(DECR *a*) - FSUBR -

identique à INCR mais décrémente de 1 la valeur numérique de l'atome littéral *a*. INCR ramène la nouvelle valeur de *a* en valeur.

ex : ? (SETQ X 10) → 10
? (DECR X) → 9
? X → 9

(NCONC *l1 l2*) - SUBR -

relie physiquement les listes *l1* et *l2* (i.e. place dans le CDR du dernier élément de la liste *l1* la liste *l2*). NCONC ramène la nouvelle liste *l1* en valeur.

ex : ? (NCONC '(A B C) '(D E F)) → (A B C D E F)

(NCONC1 *l s1 ... sN*) - NSUBR -

équivalent à (NCONC *l* (LIST *s1 ... sN*)). Cette fonction est très utile pour placer une suite d'atomes à la fin d'une autre liste.

ex : (NCONC1 '(A B C) 'D 'E) → (A B C D E)



(SMASH *l*) - SUBR -

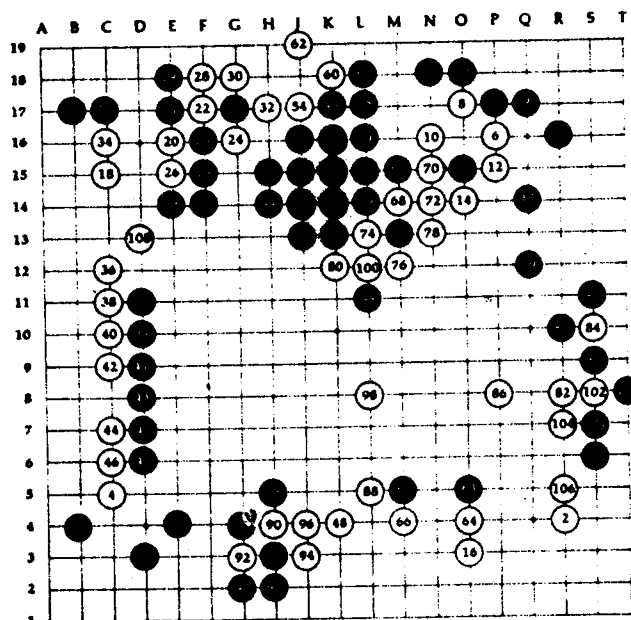
enlève physiquement le premier élément de la liste *l* en conservant la même adresse physique pour la liste *l*. Il y a donc re-copie du deuxième élément de *l* dans le premier élément puis destruction du deuxième élément. SMASH ramène la nouvelle liste *l* en valeur.

ex : ? (SETQ L1 '(A B C) L2 L1) → (A B C)
? (SMASH L2) → (B C)
? L1 → (B C)

(ATTACH *l s*) - SUBR -

ajoute physiquement en tête de la liste *l* l'élément *s* en conservant la même adresse physique pour la liste *l*. Il y a donc re-copie du premier élément de *l* dans un doublet libre puis modification de ce premier élément par *s*. ATTACH ramène la nouvelle liste *l* en valeur.

ex : ? (SETQ L1 '(Y Z) L2 L1) → (Y Z)
? (ATTACH 'X L2) → (X Y Z)
? L1 → (X Y Z)





## 2.8 FONCTIONS SUR LES A-LISTES

En VLISP-10 comme dans tous les VLISPs, les A-listes (listes d'Association) sont des listes de listes qui ont la structure suivante :

((*var1* . *val1*)(*var2* . *val2*) ... (*varN* . *valN*))

Chaque élément est une liste dont le CAR est la variable, le CDR la valeur. Pour toutes ces fonctions l'argument *al* doit être une A-liste.

### 2.8.1 Fonctions de recherche sur A-liste

(ASSQ *a al*) - SUBR -

ramène l'élément de la A-liste *al* dont le CAR est égal à l'atome *a*, sinon ramène NIL. Cette fonction utilise le prédicat EQ.

ex : ? (ASSQ 'B '((A)(B . 1)(C D E))) → (B . 1)

(CASSQ *a al*) - SUBR -

identique à ASSQ mais ramène, s'il existe, le CDR de l'élément recherché.

ex : ? (CASSQ 'C '((A)(B . 1)(C D E ))) → (D E)

(ASSOC *s al*) - SUBR -

ramène l'élément de la A-liste *al* dont le CAR est égal à l'expression *s* (atome ou liste), sinon ramène NIL. Cette fonction utilise le prédicat EQUAL.

ex : ? (ASSOC '(X Y) '((A)((X Y) Z))) → ((X Y) Z)

(CASSOC *s al*) - SUBR -

identique à ASSOC mais ramène, s'il existe, le CDR de l'élément recherché.

ex : ? (CASSOC '(X Y) '((A)((X Y) Z))) → (Z)






### 2.8.2 Fonction de création de A-liste

(PAIRLIS *l1 l2 al*) - SUBR -

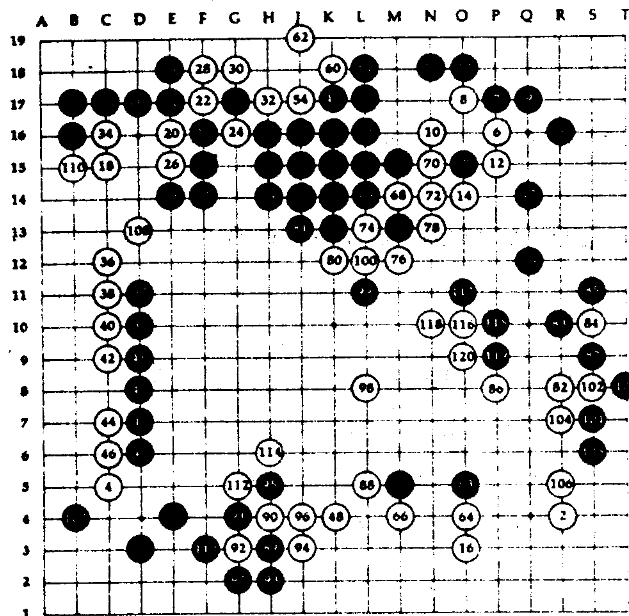
*l1* doit être une liste de variables.

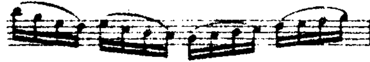
*l2* doit être une liste de valeur.

PAIRLIS ramène une nouvelle A-liste formée à partir de la liste des variables et de la liste des valeurs.

Si le troisième argument *al* est donné il est ajouté en fin de la A-liste ainsi formée.

ex :
```
? (PAIRLIS '(A B C) '(1 2 3))
→ ((A . 1)(B . 2)(C . 3))
? (PAIRLIS '(X Y Z) '(5 (6)) '((A . 1)(B . 2))
→ ((X . 5)(Y 6)(Z)(A . 1)(B . 2))
```





## 2.9 FONCTIONS SUR LES P-LISTES

En VLISP-10, comme dans tous les autres VLISPs, les P-listes (listes de Propriétés) sont des listes qui ont la structure suivante :

(*indic1 val1 indic2 val2 ... indicN valN*)

A chaque indicateur est associée une valeur. Les recherches sur P-liste s'effectuent donc deux éléments par deux éléments.

Les arguments de ces fonctions sont :

*pl* - si *pl* est un atome littéral, la P-liste utilisée sera celle de cet atome (son CDR),
- si *pl* est une liste, la P-liste utilisée sera cette liste elle-même,
- si *pl* est un nombre ou une chaîne, ces fonctions risquent de donner d'étranges résultats et de dangereuses visions ;

*ind* l'indicateur est un atome de n'importe quel type car la recherche des indicateurs utilise le prédicat EQ ;

*pval* peut être n'importe quelle expression.

### 2.9.1 Fonction de recherche sur P-liste

(GET *pl ind*) - SUBR -

ramène la valeur associée à l'indicateur *ind* dans la P-liste *pl*. Si l'indicateur n'existe pas GET ramène NIL.

Attention : on ne peut discerner la valeur égale à NIL d'un indicateur avec l'absence de cet indicateur.

ex : ? (GET '(I1 A I2 B) 'I2) → B



### 2.9.2 Fonctions de création de P-liste

(ADDPROP *pl pval ind*) - SUBR -

rajoute en tête de la P-liste *pl* l'indicateur *ind* et sa valeur associée *pval*. ADDPROP ramène *pl* en valeur.

ex : ? (ADDPROP '(I1 A I2 B) 'C 'I1) → (I1 C I1 A I2 B)

(PUT *pl pval ind*) - SUBR -

si l'indicateur *ind* existe déjà sur la P-liste *pl* alors sa nouvelle valeur est *pval* ; sinon l'indicateur *ind* et sa valeur associée *pval* sont ajoutés en tête de *pl* (d'une manière identique à la fonction ADDPROP).PUT ramène *pl* en valeur.

(REMPROP *pl ind*) - SUBR -

enlève de la P-liste *pl* l'indicateur *ind* s'il existe ainsi que sa valeur associée. REMPROP ramène *pl* en valeur.

L'utilisation conjuguée des fonctions REMPROP et ADDPROP permet d'utiliser les P-listes comme des piles de propriétés-valeurs.

ex : ? (PUT '(I1 A I2 B) 'C 'I1) → (I1 C I2 B)
? (REMPROP '(I1 A I2 B) 'I1) → (I2 B)



## 2.10 LES FONCTIONS DE DEFINITION

Elles permettent de définir des fonctions en modifiant les P-listes des atomes d'une manière plus aisée qu'en utilisant les fonctions sur P-liste.

(DE *a l s1 ... sN*) - FSUBR -

définit une fonction de type EXPR avec :

- *a* (atome littéral), comme nom de la fonction ;
- *l* (atome littéral ou liste), comme argument de la fonction ;
- et *s1 ... sN* comme corps de la fonction.

DE ramène *a* en valeur et est équivalent à

(PUT '*a* '(LAMBDA *l s1 ... sN*) EXPR)

(DF *a l s1 ... sN*) - FSUBR -

équivalent à DE, mais la fonction ainsi définie est du type FEXPR. DF ramène *a* en valeur et est équivalent à :

(PUT '*a* '(LAMBDA *l s1 ... sN*) FEXPR)

(DMI *a l s1 ... sN*) - FSUBR -

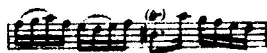
équivalent à DE, mais la fonction ainsi définie est une macro-fonction d'entrée. DMI ramène *a* en valeur et est équivalent à :

(PUT '*a* '(LAMBDA *l s1 ... sN*) MACIN)

(DMO *a l s1 ... sN*) - FSUBR -

équivalent à DE, mais la fonction ainsi définie est une macro-fonction de sortie. DMO ramène *a* en valeur et est équivalent à :

(PUT '*a* '(LAMBDA *l s1 ... sN*) MACOUT)



## 2.11 LES FONCTIONS SUR PILE

L'utilisateur a accès à une pile, de $(200)_8$ mots par défaut, au moyen des fonctions suivantes :

(PUSH *s1* ... *sN*) - NSUBR -

empile les valeurs des différentes expressions *s1* ... *sN* et ramène *sN* en valeur.

(POP) - SUBR -

dépile et ramène le sommet de pile en valeur.

Les erreurs suivantes apparaissent en cas de débordement de pile :

** USER STACK OVERFLOW. il y a eu trop de PUSH ;

** USER STACK UNDERFLOW. il y a eu trop de POP.

```
ex : ? (PUSH 'A 2) → 2
     ? (POP) → 2
     ? (POP) → A
     ? (POP)
         ** USER STACK UNDERFLOW.
     ? (WHILE T (PUSH T))
         ** USER STACK OVERFLOW
```



## 2.12 LES FONCTIONS NUMERIQUES

### 2.12.1 Les prédicats numériques

Ils servent à comparer des nombres. Tous ramènent NIL si le test effectué echoue, ou si 1 ou plusieurs arguments ne sont pas numériques.

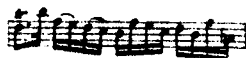
| fonction | teste si | et ramène si le test réussi |
|---|---|---|
| (NUMBP *n*) | *n* est un nombre | *n* |
| (ZEROP *n*) | $n = 0$ | *n* |
| (NEROP *n*) | $n \neq 0$ | *n* |
| (LEZP *n*) | $n \leq 0$ | *n* |
| (LZP *n*) | $n < 0$ | *n* |
| (GEZP *n*) | $n \geq 0$ | *n* |
| (GZP *n*) | $n > 0$ | *n* |
| (EVENP *n*) | *n* est pair | *n* |
| (ODDP *n*) | *n* est impair | *n* |
| (DIVP *n1 n2*) | *n1* est divisible par *n2* | *n1* |
| (LT *n1* ... *nN-1 nN*) | $n1<...<nN-1<nN$ | *nN-1* |
| (LE *n1* ... *nN-1 nN*) | $n1\leq...\leq nN-1<nN$ | *nN-1* |
| (GT *n1* ... *nN-1 nN*) | $n1>...>nN-1>nN$ | *nN-1* |
| (GE *n1* ... *nN-1 nN*) | $n1\geq...\geq nN-1\geq nN$ | *nN-1* |



## 2.12.2 Fonctions à 1 argument (SUBR)

Si l'argument n'est pas du type voulu, ces fonctions ramènent 0.

| fonction | type de l'argument | ramène le nombre |
|---|---|---|
| (LENGTH *l*) | liste | d'éléments de la liste *l* |
| (PLENGTH *a*) | atome | de caractères du P-name de *a* |
| (CASCII *c*) | caractère | correspondant au code ASCII du caractère *c* |
| (ADD1 *n*) | nombre | $n + 1$ |
| (SUB1 *n*) | nombre | $n - 1$ |
| (MINUS *n*) | nombre | $- n$ (complément arithmétique) |
| (ABS *n*) | nombre | $\|n\|$ (valeur absolue de *n*) |
| (COMPL *n*) | nombre | $\overline{n}$ (complément logique) |
| (SWAP *n*) | nombre | résultant de l'échange des 18 bits de droite avec les 18 bits de gauche de *n* |



2.12.3 Fonctions à *N* arguments (NSUBR)

Pour toutes ces fonctions les arguments non numériques sont ignorés.

| fonction | ramène |
|---|---|
| (PLUS *n1* ... *nN*) | *n1* + *n2* + ... + *nN* |
| (DIFFER *n1* ... *nN*) | *n1* - *n2* - ... - *nN* |
| (TIMES *n1* ... *nN*) | *n1* * *n2* * ... * *nN* |
| (QUO *n1* ... *nN*) | *n1* / *n2* / ... / *nN* |
| (REM *n1* ... *nN*) | *n1* REM *n2* REM ... REM *nN* (REM est l'opérateur binaire : reste de la division entière |
| (MIN *n1* ... *nN*) | le plus petit *n* |
| (MAX *n1* ... *nN*) | le plus grand *n* |

2.12.4 La fonction spéciale BOOLE

(BOOLE *numéro n1* ... *nN*) - NSUBR -

applique l'opérateur binaire booléen sélecté par *numéro* sur les différents arguments numériques *n1* ... *nN*. BOOLE ramène le nombre résultant du calcul :

(*operat* ... (*operat* (*operat n1 n2*) *n3*) ... *nN*)

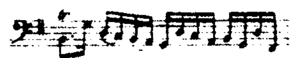
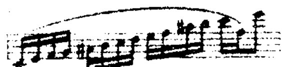


Table des opérateurs booléens :

| numéro | opérateur | |
|---|---|---|
| 0 | $n1$ | |
| 1 | $n1 \vee n2$ | union |
| 2 | $\neg n1 \vee n2$ | |
| 3 | $n1 \vee \neg n2$ | |
| 4 | $\neg n1 \vee \neg n2$ | |
| 5 | $n1 \wedge n2$ | intersection |
| 6 | $\neg n1 \wedge n2$ | |
| 7 | $n1 \wedge \neg n2$ | |
| 8 | $\neg n1 \wedge \neg n2$ | |
| 9 | $n1$ XOR $n2$ | disjonction logique |
| 10 | $n1$ EQV $n2$ | équivalence logique |
| 11 | $n1$ LOGSHIFT $n2$ | décalage logique |
| 12 | $n1$ ROTSHIFT $n2$ | décalage circulaire |
| 13 | $n1$ ARSHIFT $n2$ | décalage arithmétique |

Pour les décalages $n2$ est le nombre de pas

si $n2 > 0$ les décalages s'effectuent par la gauche,
si $n2 < 0$ les décalages s'effectuent par la droite.



## 2.13 FONCTIONS SUR LES P-NAMES DES ATOMES

Rappellons que

- le P-name d'un atome littéral est son nom externe
- le P-name d'un nombre est sa représentation dans la base de conversion courante
- le P-name d'une chaîne est la suite de caractères qui constituent cette chaîne.

**(GENSYM *a1* ... *aN*)** -NSUBR-

- si aucun argument n'est donné, GENSYM ramène à chaque appel un nouvel atome littéral de type G*n* dans lequel *n* est un nombre incrémenté à chaque appel. Pour l'initialisation de ce compteur voir la fonction 2.15.9.
- si des arguments sont donnés, GENSYM ramène un nouvel atome dont le P-name est la concaténation de tous les P-name des arguments atomiques jusqu'à concurence de 18 caractères.

ex : ? (GENSYM 'LAB (ADD1 3) ':) → LAB4:
? (GENSYM) → G101
? (GENSYM) → G102

**(EXPLODE *a1* ... *aN*)** -NSUBR-

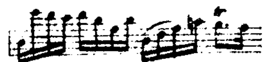
ramène la liste concaténée de tous les caractères des P-names des arguments atomiques.

ex : ? (EXPLODE 'NUM -237 'HAA) → (N U M - 2 3 7 H A A)

**(PLENGTH *a*)** -SUBR-

ramène le nombre de caractères du P-name de *a*. (voir 2.12.2)

ex : ? (PLENGTH -128) → 4
? (PLENGTH 'ATOM) → 4



(ASCII *n*)  - SUBR -

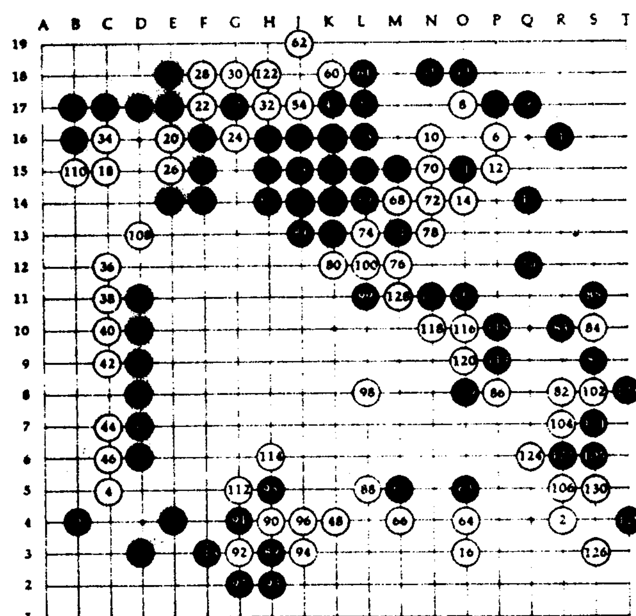
ramène le caractère de code ASCII *n* (modulo 256)

ex : ? (ASCII 67) → C

(CASCII *c*)  - SUBR -

ramène le code ASCII du caractère *c*. (voir 2.12.2)

ex : ? (CASCII 'C) → 67





## 2.14 FONCTIONS D'ENTREE/SORTIE

### 2.14.1 Les fichiers

Dans l'état actuel du système, VLISP-10 peut gérer 2 fichiers : un fichier d'entrée et un fichier de sortie. Le système multifichier est en cours de mise au point.

La description d'un *fichier* est une liste de la forme :

( device ( filename . extension ) ( pj . pg ) )

ex : ( DSK ( PROG . VLI ) ( 767 . 542 ) )

Les valeurs par défaut de ces paramètres sont :

device = TTY

filename.extension = en entrée LISPIN . VLI (VLI est l'extension standard des fichiers source en VLISP-10)

en sortie LISPOU . LST

pj.pg = numéro de projet-programmeur de l'utilisateur

(INPUT *fichier*) - SUBR -

cette fonction permet de sélecter le fichier d'entrée (après avoir fermé le fichier d'entrée précédent). A l'ouverture d'un fichier TTY, le message --- ALLO ? --- est imprimé sur cette TTY, et un ? est imprimé avant toute lecture de ligne.

(OUTPUT *fichier*) - SUBR -

sélecte le fichier de sortie (après avoir fermé le fichier de sortie précédent).

Ces deux fonctions ramènent en valeur les caractéristiques du device ouvert (la valeur du DEVCHR UUO). Pour ces deux fonctions, si aucun argument n'est donné, les fichiers standard sont ouverts. Ces fichiers standard sont en entrée et en sortie (TTY).



## 2.14.2 Erreurs pendant l'exécution des fonctions INPUT OUTPUT

*** OPEN ERROR (INPUT).

le device d'entrée est incorrect (inexistant ou impossible à lire). Le fichier d'entrée standard (TTY) est ouvert à sa place.

*** OPEN ERROR (OUTPUT).

le device de sortie est incorrect (inexistant ou impossible à écrire). Le fichier de sortie standard (TTY) est ouvert à sa place.

*** LOOKUP ERROR (INPUT) : *n*

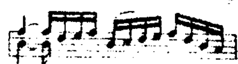
erreur à l'ouverture du fichier d'entrée. Le fichier d'entrée standard est ouvert à sa place. *n* est le type de l'erreur.

*** ENTER ERROR (OUTPUT) : *n*

erreur à l'ouverture du fichier de sortie. Le fichier de sortie standard est ouvert à sa place. *n* est le type de l'erreur.

type des erreurs les plus fréquentes

0 le fichier n'existe pas
1 le pj.pg est incorrect
2 le fichier est protégé
6 erreur physique de lecture ou d'écriture
14 il n'y a plus de place sur le disque (OUTPUT).



## 2.14.3 Fonctions d'entrée

Toutes ces fonctions utilisent le fichier d'entrée sélecté par la fonction INPUT.

(READ) - SUBR -

lit et ramène en valeur la S-expression suivante (atome ou liste) du fichier d'entrée.

(READCH) - SUBR -

lit et ramène en valeur le caractère suivant du fichier d'entrée.

(PEEKCH) - SUBR -

ramène en valeur le caractère suivant du fichier d'entrée, mais sans le lire : si (READCH) suit, le même caractère sera ramené.

(TEREAD) - SUBR -

saute à l'enregistrement suivant du fichier d'entrée (une ligne sur TTY, une carte perforée, ...).

(READSTR) - SUBR -

ramène l'enregistrement suivant sous la forme d'une chaîne de caractères (voir 3). SNOBOL 4 n'est pas le cousin de VLISP-10.

(EOF) - SUBR -

cette fonction est lancée automatiquement par le système à la fin du fichier d'entrée. Elle peut également être lancée et redéfinie par l'utilisateur.



## 2.14.4 Fonctions de sortie

Toutes écrivent sur le fichier de sortie sélecté par la fonction OUTPUT, en utilisant un buffer accessible à l'utilisateur.

(PRINT *s1* ... *sN*) - NSUBR -

édite dans le buffer de sortie les S-expressions *s1* ... *sN* puis imprime ce buffer suivi des caractères Carriage Return/Line Feed (saut de ligne). PRINT ramène *sN* en valeur.

(PRIN1 *s1* ... *sN*) - NSUBR -

édite dans le buffer de sortie les S-expressions *s1* ... *sN* sans imprimer le buffer. PRIN1 ramène *sN* en valeur.

(TERPRI *n*) - SUBR -

imprime le buffer de sortie puis saute *n* lignes. Si *n* = 0 la ligne suivante sera surimprimée. Si *n* n'est pas un nombre il est pris à 1. TERPRI ramène *n* en valeur.

(PAGE) - SUBR -

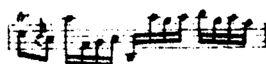
imprime le buffer de sortie puis saute une page.PAGE ramène NIL.

(PRINC *c* *n*) - SUBR -

édite *n* fois le caractère *c*. Si *n* n'est pas un nombre *c* n'est édité qu'une seule fois.PRINC ramène *c* en valeur.

(SPACES *n*) - SUBR -

édite *n* espaces. Si *n* n'est pas un nombre un seul espace sera édité. SPACES ramène *n* en valeur.



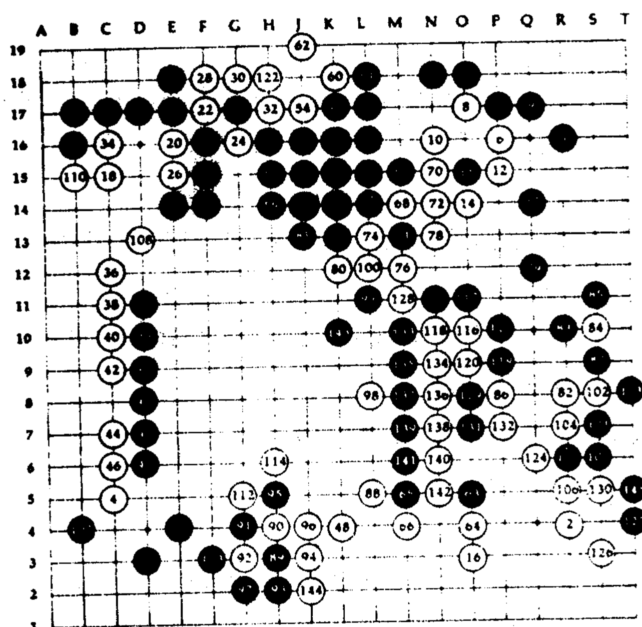
(TTAB *n*) - SUBR -

mets le pointeur du buffer de sortie à la position *n* . TTAB ramène *n* en valeur.

Remarque : Si au cours d'une édition le buffer de sortie est plein il est automatiquement imprimé, suivi d'un RC/LF et l'édition se poursuit sur la ligne suivante.

L'utilisateur a en outre accès aux indicateurs des fonctions de sortie et au buffer de sortie au moyen des fonctions

- sur le Registre Général (2.15.1)
- sur le buffer de sortie (2.15.4)





## 2.14.5 Les macro-fonctions d'entrée/sortie

Si au cours d'une lecture la fonction READ lit une liste dont le CAR, atomique, contient sur sa P-liste l'indicateur MACIN, la fonction associée à cet indicateur est lancée avec comme argument le CDR de la liste qui a été lue : la valeur retournée par cette fonction remplace la liste lue. Les macro-fonctions d'entrée sont définies au moyen de la fonction DMI. L'action des macro-fonctions d'entrée est validée par le bit 16 du R.G. (voir 2.15.1).

De même si au cours d'une édition les fonctions PRINT ou PRIN1 doivent éditer une liste dont le CAR, atomique, contient sur sa P-liste l'indicateur MACOUT, la fonction associée à cet indicateur est lancée avec pour argument le CDR de la liste qui devait être éditée. Les macro-fonctions de sortie sont définies au moyen de la fonction DMO.

L'action des macro-fonctions de sortie est validée par le bit 26 du R.G. (voir 2.15.1).

ex 1 : si la fonction IF n'était pas standard, on pourrait la macrogénérer de la manière suivante

```
(DMI IF (B THEN . ELSE)
    (LIST 'COND (LIST B THEN)(CONS T ELSE)))
```

ainsi la liste lue

```
(IF (GT A B)(SETQ X B)(SETQ X A)(INCR Y))
```

serait transformée en

```
(COND ((GT A B)(SETQ X B))
      (T (SETQ X A)(INCR Y)))
```

ex 2 : si la restitution du macro-caractère ', en sortie, n'était pas standard on pourrait la définir ainsi

```
(DMO QUOTE X (PRIN1 (QUOTE ') X))
```



## 2.15 LES FONCTIONS MAGIQUES STATUS

Les fonctions STATUS permettent tout ce **que vous avez toujours** eu envie de faire à un système LISP sans jamais pouvoir le faire vraiment. D'une manière plus précise, les fonctions STATUS permettent de {connaître / modifier} les valeurs des {indicateurs / variables} internes du système VLISP-10.

Syntaxe : (STATUS *numéro arg1 ... argN*) - NSUBR -

Le *numéro* du STATUS permet de sélecter la fonction à exécuter. Dans l'état actuel du système 40 numéros sont prévus. Un *numéro* de STATUS incorrect déclenche une erreur avec impression du libellé :

** STATUS ERROR : *numéro incorrect*

Les fonctions STATUS sont les seules qui contrôlent le type et la valeur de *tous* leurs arguments.

Il existe pour chacune de ces fonctions un mnémonique. Pour pouvoir les utiliser il faut au préalable charger le fichier :

(DSK (STATUS . VLI)\(630 . 300))

ex : (SETBIT 20 22)

est équivalent à :

(STATUS 1 20 22)



### 2.15.1 Le Registre Général R.G.

Il existe un registre de 36 bits qui contient la plupart des indicateurs du système : le R.G. Ses différents bits ont la signification suivante :

| bit | val/ def. | si le bit est à 1 |
|---|---|---|
| 1 | 1 | imprime le résultat du top-level |
| 2 | 0 | imprime les formes à évaluer |
| 3 | 0 | trace tous les appels internes de la fonction EVAL |
| 4 | 0 | trace tous les appels internes de la fonction APPLY |
| 5 | 0 | donne des statistiques à chaque Garbage Collecting |
| ............... | | |
| 10 | 0 | tous les enregistrements lus en entrée sont imprimés |
| 11 | 1/0 | le fichier d'entrée est un fichier TTY. Il y a impression du caractère ? avant chaque lecture. Ce bit est positionné par la fonction INPUT |
| 12 | 0 | en entrée un nombre peut commencer par le caractère + |
| 13 | 1 | en entrée un nombre peut commencer par le caractère - |
| 14 | 1 | en entrée le quote caractère / est valide |
| 15 | 1 | en entrée les macro-caractères sont traités |
| 16 | 1 | en entrée les macro-fonctions sont traitées |
| 17 | 1 | en entrée les chaînes de caractères sont acceptées |
| 18 | 1 | en entrée les commentaires sont acceptés |
| ............... | | |
| 20 | 1 | les impressions physiques ont lieu à la fin de chaque ligne (et non à la fin du buffer système) |
| 21 | 1 | toutes les éditions commencent par un espace |
| 22 | 0 | en sortie les nombres positifs commencent par le caractère + |
| 23 | 1 | en sortie les nombres négatifs commencent par le caractère - |
| 24 | 0 | en sortie les caractères spéciaux des P-names sont quotés par / |
| 25 | 1 | en sortie on met un espace entre chaque atome |
| 26 | 1 | en sortie les macro-fonctions sont traitées |
| 27 | 1 | en sortie le caractère délimiteur de chaîne est restitué |
| ............... | | |
| 35 | 0 | réalise la fonction QUASAR. Ne l'utilisez jamais! |



## 2.15.2 Fonctions sur le R.G.

**(STATUS 0 *n*)** - mnémo : SETRG -

si *n* est un nombre, donne au R.G. cette nouvelle valeur. Ce status ramène le nombre qui représente la valeur du R.G. courant.

**(STATUS 1 *n1* ... *nN*)** - mnémo : SETBIT -

met à 1 tous les bits du R.G. de numéro *n1* ... *nN*. Si un de ces arguments n'est pas un nombre tel que $0<n_i<35$, l'erreur status est déclenchée. Ce status ramène *nN*.

**(STATUS 2 *n1* ... *nN*)** - mnémo : CLRBIT -

identique à (STATUS 1 ... ) en mettant les bits à 0.

**(STATUS 3 *n1* ... *nN*)** - mnémo : NEGBIT -

identique à (STATUS 1 ... ) en inversant les bits.

**(STATUS 4 *n1* ... *nN*)** - mnémo : TESBIT -

teste des bits du R.G. Ce status ramène *nN* si tous les bits de numéro *n1* ... *nN* sont positionnés à 1, sinon ramène NIL.

## 2.15.3 Conversion des nombres

Le traitement des signes est contrôlé :

- en entrée par les bits 12 et 13 du R.G. ;
- en sortie par les bits 22 et 23 du R.G.

On peut également modifier les bases de conversion des nombres qui sont, par défaut, décimales.



(STATUS 5 *n*) - mnémo : IBASE -

si 2<*n*<16, *n* est la nouvelle base de conversion des nombres en entrée. Ce status ramène la base de conversion courante. Il existe un macro-caractère standard, l'antislash \ , qui permet de lire temporairement (le temps d'une S-expression) des nombres en octal.

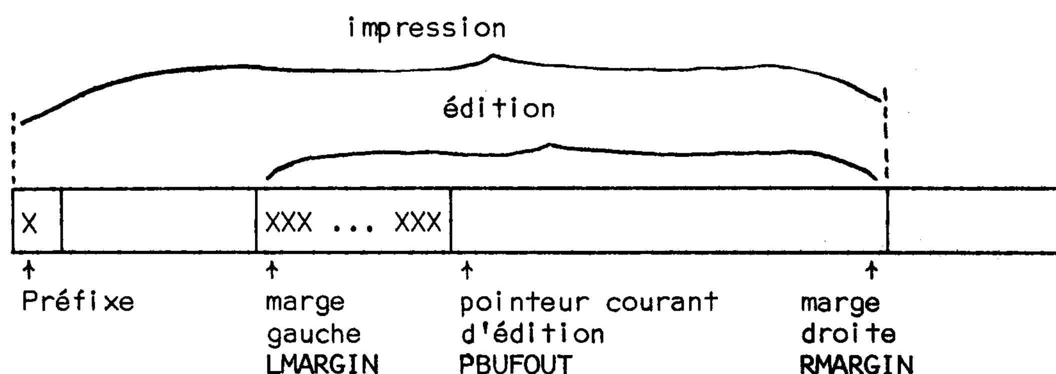
ex : \(630 . 300) est équivalent à (408 . 192)

(STATUS 6 *n*) - mnémo : OBASE -

si 2<*n*<16, *n* est la nouvelle base de conversion des nombres en sortie. Ce status ramène la base de conversion courante.

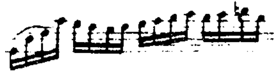
## 2.15.4 Le buffer de sortie

Le buffer de sortie est organisé de la manière suivante :




(STATUS 7 *n*) - mnémo : LMARGIN -

si *n* est un nombre, donne la valeur *n* à la marge gauche. Si RMARGIN<*n*<0 l'erreur status se déclenche. Ce status ramène la valeur courante de la marge gauche.



(STATUS 8 *n*) - mnémo : PBUFOUT -

si *n* est un nombre,donne la valeur *n* au pointeur courant d'édition. Si RMARGIN<*n*<0 l'erreur status se déclenche. Ce status ramène la valeur courante du pointeur d'édition.

(STATUS 9 *n*) - mnémo : RMARGIN -

si *n* est un nombre,donne la valeur *n* à la marge droite. si 118<*n*<LMARGIN l'erreur status se déclenche. Ce status ramène la valeur courante de la marge droite.

Si les fonctions SPACES et TTAB n'étaient pas standard,on pourrait les définir de la manière suivante :

```
(DE SPACES (N ;; AUX)
    (SETQ AUX (PLUS (STATUS 8) N))
    (IF (LT AUX (STATUS 9))(STATUS 8 AUX)(TERPRI)))

(DE TTAB (N)
    (AND (GEZP N)(LT N (STATUS 9))(STATUS 8 N)))
```

## 2.15.5 Les préfixes

A chaque impression, un caractère préfixe est ajouté à gauche de la ligne, pour différencier les S-expressions objets ou résultats.
Pour les 3 status ci-dessous, l'argument, s'il est donné, doit être un caractère (c'est-à-dire un atome littéral mono-caractère).

(STATUS 11 *c*) - mnémo : PREFORM -

ce status permet de modifier le caractère préfixe de toutes les formes lues par la fonction TOPLEVEL, en vue de leurs évaluations. Ce caractère est également envoyé sur la TTY avant toute lecture de ligne. Par défaut ce caractère est le ?.



(STATUS 12 *c*) - mnémo : PREFTOPL -

ce status permet de modifier le caractère préfixe du résultat de l'évaluation de la fonction TOPLEVEL. Par défaut ce caratère est le =.

(STATUS 13 *c*) - mnémo : PREFPRINT -

ce status permet de modifier le caractère préfixe de toutes les impressions de l'utilisateur. Par défaut ce caractère est l'espace n.

2.15.6 Les caractères spéciaux

L'analyseur lexical VLISP (la fonction READ) exécute un traitement particulier pour les caractères suivants :

- le quote caractère, par défaut /. Il permet d'insérer des séparateurs dans le P-name d'un atome. Son action est validée par le bit 14 du R.G. ;
- le délimiteur de commentaires, par défaut ;. Tous les caractères qui se trouvent entre 2 ; seront ignorés à la lecture. Son action est validée par le bit 18 du R.G. ;
- le délimiteur de chaînes de caractères, par défaut ". Son action est validée par le bit 17 du R.G..

Certains status permettent de modifier et/ou de connaître ces caractères spéciaux : leur argument *c* doit être un caractère (atome mono-caractère) sous peine de déclencher l'erreur status.

(STATUS 14 *c*) - mnémo : CQUOTE -

Si *c* est donné, modifie le caractère spécial quote caractère. Ce status ramène le caractère quote-caractère courant.



(STATUS 15 *c*)   - mnémo : CCOMMENT -

si *c* est donné, modifie le caractère spécial délimiteur de commentaire. Ce **status** ramène le caractère délimiteur de commentaires courant.

(STATUS 16 *c*)   - mnémo : CSTRING -

modifie le caractère spécial délimiteur de chaîne. Si *c* est donné, ce status ramène le caractère délimiteur de chaînes cuo courant.

ex : ? (STATUS 14) → /
? (STATUS 15) → ;
? (STATUS 16) → "

Tous les autres caractères ont un type qui peut être :
0 séparateur d'enregistrement (ex : RC LF escape ...)
1 séparateurs nuls (ex : espace tab ... )
2 caractère normal
3 .
4 (
5 )

(STATUS 17 *c n*)   - mnémo : TYPECHAR -

permet de modifier le type du caractère *c* par le nombre *n* si *n* est un nombre. Si *c* n'est pas un caractère ou si *n* n'est pas un nombre tel que 0<*n*<5, l'erreur status est déclenchée. Ce status ramène la valeur du type courant du caractère *c* ; il permet de changer les tables de l'analyseur lexical VLISP-10.

ex : ? (STATUS 17 '/.) → 3
? (STATUS 17 '/< 4) → 4
? (STATUS 17 '/> 5) → 5

l'entrée   <CONS '(A) '<B>>
sera comprise comme   (CONS '(A) '(B))



## 2.15.7 Les macro-caractères

Tous les caractères normaux (de type 2) peuvent servir de macro-caractère. Un macro-caractère est un caractère auquel est associé une fonction qui est lancée automatiquement, avec NIL comme argument, à chaque lecture de ce caractère. La valeur ramenée par cette fonction remplace le macro-caractère lu. Cette action est validée par le bit 15 du R.G.

Il existe 2 macro-caractères standard :

- le quote ' placé devant une S-expression quelconque ramène la liste (QUOTE S-expression)

  ex : '(A B) est équivalent à (QUOTE (A B))
  
  ''A " " " (QUOTE (QUOTE A))

- l'anti-slash \ placé devant une S-expression ramène cette S-expression en considérant que tous les nombres qui y sont inclus sont écrits en octal.

  ex : \(630 . 300) est équivalent à(408 . 192) en décimal

(STATUS 18 *c s*) - mnémo : MACHAR -

permet de donner la définition *s* (si *s* est donné) de la fonction associée au macro-caractère *c*. Ce status ramène la définition courante du macro-caractère *c*. Si *c* n'est pas un caractère (atome mono-caractère) l'erreur status se déclenche.

(STATUS 19 *c*) - mnémo : SELCHAR -

permet de détruire la définition du macro-caractère *c*. *c* n'est donc plus un macro-caractère. Si *c* n'est pas un caractère (atome mono-caractère), l'erreur status se déclenche.



(STATUS 20) - mnémo : LASTREAD -

ce dernier status ramène un pointeur sur **la dernière S-expression** lue avant l'appel du macro-caractère courant. **Ce status est** utilisé pour connaître **(CAR (STATUS 20)) ou pour modifier** (SET (STATUS 20) ...)la dernière S-expression **lue juste avant** l'appel d'un macro-caractère.

ex 1 :si les macro-standard n'étaient pas **définies, on pourrait** le faire de la manière suivante

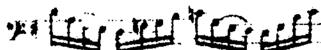
```
(STATUS 18 (QUOTE /')(QUOTE (LAMBDA ()
      (LIST QUOTE (READ)))))

(STATUS 18 '\    '(LAMBDA (L)
    (PUSH (STATUS 5))
    (STATUS 5 8)
    (SETQ L (READ))
    (STATUS 5 (POP))
    L))
```

ex 2 : création de 2 macro-caractères : **pour CONS et := pour le** SETQ en notation infixé :

```
(STATUS 18 '/: '(LAMBDA ()
    (PUSH (CAR (STATUS 20)))
    (SET (STATUS 20)
       (IF (NEQ (PEEKCH)'/=) 'CONS
           (READCH) 'SETQ))
    (POP))
```

| | | | |
|---|---|---|---|
| la lecture de | (A : B) | sera équivalente à | **(CONS A B)** |
| " " | " (I := 4) | " " " | **(SETQ I 4)** |



## 2.15.8 Le Garbage Collecting

Le Garbage-Collecting (G.C.) récupère les doublets libres de la liste libre (si celle-ci vient à épuisement), ainsi que les atomes, non utilisés. Si un G.C. n'a pu récupérer des doublets, le message d'erreur suivant apparaît :

** FULL MEMORY.

Certains status permettent de contrôller le travail du G.C.

Le bit 5 du R.G.

permet d'obtenir une trace et des statistiques à chaque G.C.

(STATUS 21) - mnémo : GARBCOLL -

lance un G.C. et ramène en valeur le nombre de doublets libérés.

(STATUS 22) - mnémo : LFREE -

ramène le nombre de doublets libres (sans lancer de G.C.)

(STATUS 23 *n*) - mnémo : STEPGC -

donne une valeur *n* au nombre de G.C. que l'interprète pourra effectuer avant de déclencher l'erreur libellée

** ER G.C. STEP DONE.

(STATUS 24 *n*) - mnémo : LIMGC -

donne le nombre minimum *n* de doublets récupérables à chaque G.C. Si ce nombre n'est pas atteint il y a impression du message d'avertissement suivant :

** LEFT CELLS : *n*

où *n* est le nombre de doublets réellement récupérés.





2.15.9 Compteur de GENSYM

(STATUS 26 *n*) - mnémo : GENSYMC -

donne la valeur *n* (si *n* est un nombre) au compteur de la fonction GENSYM. Ce status ramène le compteur courant en valeur.

ex : ? (GENSYM) → G101

? (STATUS 26 1000) → 1000

? (GENSYM) → G1001

2.15.10 Trace des fonctions

Le bit 3 du R.G.

permet de tracer tous les appels *internes* de la fonction interprète EVAL.

Le bit 4 du R.G.

permet de tracer tous les appels *internes* de la fonction interprète APPLY.

(STATUS 28 *a1* ... *aN*) - mnémo : TRACES -

trace les appels des fonctions standard *a1* ... *aN* ainsi que les valeurs retournées par ces fonctions, qui doivent être de type SUBR ou FSUBR.

(STATUS 29 *a1* ... *aN*) - mnémo : UNTRACES -

enlève la trace des fonctions standard *a1* ... *aN* mise par TRACES.





Exemple de TRACE

```
?   (INPUT '(DSK (TRACES . VLI)))
=   17324366083

?   (DE FACT (N) (COND ((ZEROP N) 1) ((TIMES N (FACT (SUB1 N))))))
=   FACT

?   (FACT 3)
=   6

?   (STATUS 28 'COND 'ZEROP 'TIMES 'SUB1)
=   28

?   (FACT 3)
   ---> COND :   (((ZEROP N) 1) ((TIMES N (FACT (SUB1 N)))))
   ---> ZEROP :   (3)
   <--- ZEROP =   NIL
   ---> SUB1 :   (3)
   <--- SUB1 =   2
   ---> COND :   (((ZEROP N) 1) ((TIMES N (FACT (SUB1 N)))))
   ---> ZEROP :   (2)
   <--- ZEROP =   NIL
   ---> SUB1 :   (2)
   <--- SUB1 =   1
   ---> COND :   (((ZEROP N) 1) ((TIMES N (FACT (SUB1 N)))))
   ---> ZEROP :   (1)
   <--- ZEROP =   NIL
   ---> SUB1 :   (1)
   <--- SUB1 =   0
   ---> COND :   (((ZEROP N) 1) ((TIMES N (FACT (SUB1 N)))))
   ---> ZEROP :   (0)
   <--- ZEROP =   0
   <--- COND =   1
   ---> TIMES :   (1 1)
   <--- TIMES =   1
   <--- COND =   1
   ---> TIMES :   (2 1)
   <--- TIMES =   2
   <--- COND =   2
   ---> TIMES :   (3 2)
   <--- TIMES =   6
   <--- COND =   6
=   6

?   (STATUS 29 'COND 'ZEROP 'TIMES 'SUB1)
=   29

   ** E.O.F.

?   (STOP)
```

## 2.15.11 Fonctions qui utilisent des UUOses

(STATUS 31) - mnémo : SWITCH -

ramène la valeur des clés du pupitre opérateur (SWITCH UUO).

(STATUS 32 *n*) - mnémo : LIGHTS -

affiche la valeur *n* sur les voyants du pupitre (LIGHTS UUO).

(STATUS 34 *n1 n2*) - mnémo : GETTAB -

ramène la valeur de la GETTAB UUO (accès aux tables du moniteur) avec *n1* comme index et *n2* comme numéro dans la table.

(STATUS 35) - mnémo : RUNTIME -

ramène le temps utilisé en milli-secondes depuis le début du travail (RUNTIME UUO).

(STATUS 36) - mnémo : DAYTIME -

ramène l'heure du jour en milli-secondes (DAYTIME UUO).

(STATUS 37) - mnémo : DATE -

ramène la date du jour sous la forme :

((année - 1964) × 12 + (mois - 1)) × 31 + jour - 1. (DATE UUO).

(STATUS 38) - mnémo : PJOB -

ramène le numéro du job (PJOB UUO).

(STATUS 39) - mnémo : GETPPN -

ramène le numéro de projet-programmeur utilisateur sous la forme d'une paire pointée numérique (pj.pg) (GETPPN UUO).



## 2.15.12 Récapitulatif des fonctions status

| | mnémonique | val. / def. | teste si |
|---|---|---|---|
| (STATUS 0 *n*) | SETRG | $(1655760002)_8$ | *n* >0 |
| (STATUS 1 *n1* ... *nN*) | SETBIT | | $0 \leq n_i \leq 35$ |
| (STATUS 2 *n1* ... *nN*) | CLRBIT | | $0 \leq n_i \leq 35$ |
| (STATUS 3 *n1* ... *nN*) | NEGBIT | | $0 \leq n_i \leq 35$ |
| (STATUS 4 *n1* ... *nN*) | TESBIT | | $0 \leq n_i \leq 35$ |
| (STATUS 5 *n*) | IBASE | $10_{10}$ | $2 \leq n \leq 16$ |
| (STATUS 6 *n*) | OBASE | $10_{10}$ | $2 \leq n \leq 16$ |
| (STATUS 7 *n*) | LMARGIN | 0 | $0 \leq n \leq 118$ |
| (STATUS 8 *n*) | PBUFOUT | | $0 \leq n \leq 118$ |
| (STATUS 9 *n*) | RMARGIN | $68_{10}$ | $18 \leq n \leq 118$ |
| (STATUS 11 *c*) | PREFORM | ? | *c* est un caractère |
| (STATUS 12 *c*) | PREFTOPL | = | *c* est un caractère |
| (STATUS 13 *c*) | PREFPRINT | n | *c* est un caractère |
| (STATUS 14 *c*) | CQUOTE | / | *c* est un caractère |
| (STATUS 15 *c*) | CCOMMENT | ; | *c* est un caractère |
| (STATUS 16 *c*) | CSTRING | " | *c* est un caractère |
| (STATUS 17 *c n*) | TYPECHAR | | *c* est un caractère, $0 \leq n \leq 5$ |
| (STATUS 18 *c s*) | MACHAR | | *c* est un caractère |
| (STATUS 19 *c*) | DELCHAR | | *c* est un caractère |
| (STATUS 20) | LASTREAD | | |
| (STATUS 21) | GARBCOLL | | |
| (STATUS 22) | LFREE | | |
| (STATUS 23 *n*) | STEPGC | | *n* est un nombre |
| (STATUS 24 *n*) | LIMGC | | *n* est un nombre |
| (STATUS 26 *n*) | GENSYMC | $100_{10}$ | *n* est un nombre |
| (STATUS 28 *a1* ... *aN*) | TRACES | | $a_i$ est un atome littéral |
| (STATUS 29 *a1* ... *aN*) | UNTRACES | | $a_i$ est un atome littéral |
| (STATUS 31) | SWITCH | | |
| (STATUS 32 *n*) | LIGHTS | | *n* est un nombre |
| (STATUS 34 *n1 n2*) | GETTAB | | *n1* et *n2* sont des nombres |
| (STATUS 35) | RUNTIME | | |
| (STATUS 36) | DAYTIME | | |
| (STATUS 37) | DATE | | |
| (STATUS 38) | PJOB | | |
| (STATUS 39) | GETPPN | | |



## 2.16 Les fonctions système

(LOC *s*) - SUBR -

ramène l'adresse physique de la S-expression *s*.

(VAG *n*) - SUBR -

ramène l'objet LISP qui est à l'adresse physique *n*.

ex :
```
(LOC 'X) → 846
(VAG 846) → X
(VAG (LOC '(A B C))) → (A B C)
```

(TOPLEVEL) - SUBR -

Cette fonction est appelée par la boucle principale de l'interprète. Elle peut être également appelée et redéfinie par l'utilisateur. D'une manière standard TOPLEVEL réalise des opérations semblables à

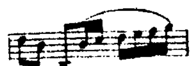
```
(DE TOPLEVEL (;;LU)
    (SETQ LU (READ))
    (AND (STATUS 4 2) (PRINT LU))
    (SETQ LU (EVAL LU))
    (AND (STATUS 4 1) (PRINT LU)))
```

Remarque : une redéfinition de cette fonction, qui n'évalue pas des S-expressions lues, rend le système inutilisable

ex : définition à éviter
```
(DE TOPLEVEL ()
    (READ) 'NAN)
```

(RESET) - SUBR -

arrête l'évaluation en cours, lance un G.C. et redonne le contrôle à la boucle principale de l'interprète.





(ERROR *s1* ... *sN*) - NSUBR -

évalue les différentes expressions *s1* ... *sN* en séquence et imprime sur le fichier standard de sortie le message :

```
** USER ERROR : sN
```

Le contrôle est ensuite rendu à la boucle principale de l'interprète.

(BREAK *s1* ... *sN*) - NSUBR -

évalue les différentes expressions *s1* ... *sN* en séquence. La valeur de la dernière évaluation i.e. celle de *sN* est la valeur du BREAK.BREAK suspend l'évaluation en cours,imprime le message

```
** ENTER BREAK : valeur du BREAK
```

et appelle la fonction TOPLEVEL tant que la valeur ramenée par cette fonction est différente de la valeur du BREAK. A la sortie d'un BREAK, le message suivant est imprimé :

```
** EXIT BREAK : valeur du BREAK
```

et l'évaluation suspendue reprend.

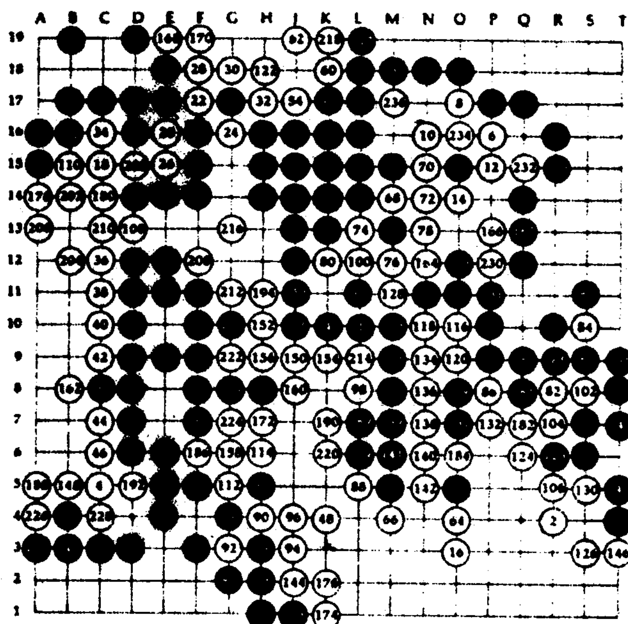
ex :

```
? (PROGN (SETQ I 10)
?        (BREAK 'RE)
?        (PRINT I))
  ** ENTER BREAK : RE
? I
= 10
? (SETQ I 11)
= 11
? 'RE
= RE
  ** EXIT BREAK : RE
  11
= 11
?
```



(STOP) - SUBR -

arrête l'évaluation en cours, ferme tous les fichiers ouverts, sort de l'interprète et rend le contrôle au moniteur (EXIT UUO).




Les Noirs 201 et 203 (B-13 et C-13) ont été pris par le coup du Blanc 206, et le Blanc 210 a été placé au C-13, occupé autrefois par le Noir 203.

Les Blancs 196 et 198 (F-10 et E-10) ont été pris par le coup du Noir 221, et le Noir 223 a été placé au F-10, occupé autrefois par le Blanc 196.



## 3 LES CHAINES DE CARACTERES

VLISP-10 permet la manipulation d'objets de type chaîne de caractères (STRING). Une chaîne de caractères est une suite illimitée, si ce n'est par la taille de la mémoire, de caractères quelconques. Sa représentation externe est cette suite de caractères encadrée du caractère délimiteur de chaîne . Si ce caractère doit être inséré dans une chaîne, il faut le doubler. Les caractéristiques d'une chaîne sont :

- les caractères qu'elle contient ;
- le nombre de ces caractères (sa longueur).

ex :

| la chaîne | contient les caractères | et a pour longueur |
|---|---|---|
| "CHAINE 10" | CHAINE 10 | 10 |
| "(""NON""?)" | ("NON"?) | 8 |
| """""" | "" | 1 |
| """" | aucun<br>cette chaîne s'appelle la chaîne vide ou nulle | 0 |

En entrée, la lecture des chaînes, par la fonction READ, est validée par le bit 17 du R.G., les chaînes sont lues en représentation externe.

En sortie, la restitution du caractère délimiteur de chaîne est validée par le bit 27 du R.G.

Le caractère délimiteur de chaîne par défaut " peut être modifié grâce à la fonction (STATUS 16 ...) (voir 2.15.6).





## 3.1 LES FONCTIONS SUR LES CHAINES

La valeur d'une chaîne de caractères est cette chaîne elle-même. Tout comme les nombres, une chaîne n'a ni C-valeur, ni P-liste. Pour toutes les fonctions qui vont suivre, les arguments, qui doivent être de type chaîne, sont symbolisés par *str* . Si ces arguments ne sont pas de type chaîne, ils sont convertis automatiquement en chaîne au moyen de la fonction **STRING**. Pour avoir accès à l'ensemble des fonctions sur les chaînes, il faut charger le fichier **(STRING . VLI)**

### 3.1.1 Les fonctions de conversion de chaînes

**(STRING *s*)** **- SUBR -**

convertit la S-expression *s* (de type quelconque) en une chaîne de caractères :

si *s* est une chaîne STRINGramène cette chaîne ;

si *s* = NIL, STRING ramène la chaîne vide ""

si *s* est un atome,STRING ramène la chaîne contenant tous les caractères du P-name de cet atome

si *s* est une liste, elle est supposée être une liste de de caractères (atomes mono-caractères). STRING ramène la chaîne composée de ces caractères.

**(MAKLIST *str*)** **- SUBR -**

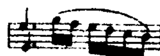
convertit la chaîne *str* en une liste de caractères (atomes mono-caractère). Si *str* est la chaîne vide,MAKLIST ramène NIL

ex : ? (STRING 'ATOME) → "ATOME"

? (STRING '(A Z T /. Q)) →"AZT.Q"

? (MAKLIST "STRG") → (S T R G)

? (MAKLIST (STRING '(A Z T /. Q))) → (A Z T /. Q)



## 3.1.2 Prédicats sur les chaînes

(STRINGP *s*) - SUBR -

ramène *s* si *s* est une chaîne, sinon ramène NIL.

(NULLSTRP *str*) - SUBR -

ramène *str* si *str* est la chaîne vide "", sinon ramène NIL.

(EQSTRING *str1 str2*) - SUBR -

ramène *str1* si la chaîne *str1* contient les mêmes caractères et a la même longueur que la chaîne *str2*. Cette fonction est équivalente à EQ, mais convertit ses 2 arguments en chaînes.

ex : (EQSTRING 'STRG "STRG") → "STRG"

## 3.1.3 Les fonctions de recherche dans les chaînes

(STRINGL *str*) - SUBR -

ramène le nombre de caractères de la chaîne *str*.

(INDEX *str1 str2 n*) - SUBR -

recherche dans la chaîne *str1* la sous-chaîne *str2* à partir du *n*ième caractère de *str1*. INDEX ramène la position du début de de la sous-chaîne *str2* par rapport au début de la chaîne *str1* Si *n* n'est pas un nombre ou est omis, la recherche s'effectue à la position 1. Si la sous-chaîne *str2* n'existe pas dans la chaîne *str1*, INDEX ramène NIL

ex : ? (INDEX "ABRACADABRA" "RAC") → 3

? (INDEX "ABRACADABRA" "RAC" 4) → NIL



## 3.1.4 Les fonctions de création de chaînes

(CONCAT *str1* ... *strN*) - SUBR -

ramène une chaîne résultant de la concaténation des copies de toutes les chaînes *str1* ... *strN*.

ex : ? (CONCAT "AB" (ADD1 14) "" 'AT) → "AB15AT"

(SUBSTRING *str n1 n2*) - SUBR -

ramène une copie de la sous-chaîne de *str* commençant à la position *n1* et se terminant à la position *n2*.

si *n1* > *n2*, SUBSTRING ramène la chaîne vide ""

si *n1* n'est pas un nombre, la sous-chaîne commence à la position 1 de la chaîne *str*

si *n2* n'est pas un nombre ou est omis, la sous-chaîne se termine à la fin de la chaîne *str*.

ex : ? (SUBSTRING "ABRACAD" 2 4) → "BRA"

? (SUBSTRING "ABRACAD" NIL 4) → "ABRA"

? (SUBSTRING "ABRACAD" 2) → "BRACAD"

(DUPL *str n*) - SUBR -

ramène une chaîne résultant de la duplication *n* fois de la chaîne *str*. Si *n* n'est pas un nombre, ou est omis, DUPL ramène une copie de la chaîne *str*. Si *n* ≤ 0, DUPL ramène la chaîne vide

ex : ? (DUPL "ALO" 3) → "ALOALOALO"

(REVERSTR *str*) - SUBR -

ramène une copie de la chaîne *str* inversée.

ex : ? (REVERSTR "ALO") → "OLA"



(REPLACE *str1 str2 str3*) - SUBR -

ramène une copie de la chaîne *str1* en y remplaçant toutes les occurences de la sous-chaîne *str2* par la chaîne *str3*.

ex :? (REPLACE "STRG1 STRG2" "STRG" "CHAINE") → "CHAINE1 CHAINE2"

(TRANSLATE *str1 str2 str3*) - SUBR -

ramène une copie de la chaîne *str1* en y remplaçant les caractères qui font partie de la chaîne *str2* par leurs homologues dans la chaîne *str3*. Si des caractères de *str2* n'ont pas d'homologues dans la chaîne *str3* (i.e. si

(STRINGL *str2*) > (STRINGL *str3*))

ils sont enlevés de la chaîne résultante.

ex :? (TRANSLATE "ABRACADABRA" "ARC" "OL") → "OBLOODOBLO"





## 3.2 EXEMPLE DE FONCTIONS SUR LES CHAINES

```
? (DE CONVBDC (NB)
?     (IF (LE NB 1)
?         (STRING NB)
?         (CONCAT (CONVBDC (QUO NB 2))
?                 (REM NB 2))))
= CONVBDC

? (CONVBDC 13)
= "1101"

? (CONVBDC 27)
= "11011"

? (DE PALINDROM (CH)
?     (SETQ CH (TRANSLATE CH ",;.:!?'()[] "))
?     (EQSTRING (REVERSTR CH) CH))
= PALINDROM

? (PALINDROM "MADAM")
= "MADAM"

? (PALINDROM "ELU PAR CETTE CRAPULE ... ")
= "ELUPARCETTECRAPULE"

? (PALINDROM "POURQUOI DES CANONS ?")
= NIL
```



## 4 - LES EDITEURS, LES CORRECTEURS

### 4.1 LE PRETTY-PRINT

Les fonctions standard PRINT et PRIN1 peuvent éditer n'importe quelle S-expression. Les seules mesures prises pour améliorer la lisibilité sont :

- l'insertion d'un espace entre chaque atome ;
- l'interdiction d'éditer un atome sur deux lignes.

Ces mesures sont nettement insuffisantes pour éditer des programmes VLISPs. Une fonction utilisateur,éditée par ces fonctions standard, devient illisible dès qu'elle dépasse quelques lignes.

Les fonctions du PRETTY-PRINT vont éditer les fonctions utilisateur d'une manière beaucoup plus lisible en faisant ressortir,au moyen de renfoncements gauches et de sauts de lignes appropriés, la structure de contrôle du programme. Les fonctions éditées au moyen du PRETTY-PRINT pourront être relues par la fonction standard de lecture : READ.

Pour avoir accès à l'ensemble des fonctions qui vont être décrites,il faut au préalable charger le fichier (DSK (PRETTY . VLI)\(630 . 300))

4.1.1. Fonctions du PRETTY-PRINT

(PRETTYP *s*)

édite la S-expression *s* en utilisant le PRETTY-PRINT. **PRETTYP** ramène *s* en valeur.

(PRETTY *a1* ... *aN*)

édite les fonctions utilisateur *a1* ... *aN* sous la forme de définitions **DE** ou **DF** en utilisant le PRETTY-PRINT. **PRETTY** ramène NIL.

(PRETTYFILE *filout filin1* ... *filinN*)

édite dans le fichier de sortie *filout* les différents fichiers *filin1* ... *filinN* en utilisant le PRETTY-PRINT. **PRETTYFILE** édite également les commentaires qui se trouvent dans les fichiers d'entrée et fait un saut de page sur le fichier de sortie à la rencontre d'un commentaire de la forme **;PAGE;** **PRETTYFILE** ramène NIL en valeur.



4.1.2 Exemple d'utilisation du PRETTY PRINT

```
=  (LOAD : PRETTY PRETTYP PRETTYFILE)

  **  E.O.F.

?  (CDR 'PRETTY)
=  (FEXPR (LAMBDA (%L %X) (MAPC %L '(LAMBDA (%L) (COND ((SETQ %X (GET
= %L EXPR)) (PRETTYP (CONS 'DE (CONS %L (CDR %X))))) ((SETQ %X (GET
= %L FEXPR)) (PRETTYP (CONS 'DF (CONS %L (CDR %X))))) ((PRETTYP %L)))
= (TERPRI)))))

?  (PRETTYP (CDR 'PRETTY))

   (FEXPR
     (LAMBDA (%L %X)
        (MAPC %L
           '(LAMBDA (%L)
              (COND
                 ((SETQ %X (GET %L EXPR))
                    (PRETTYP (CONS 'DE (CONS %L (CDR %X)))))
                 ((SETQ %X (GET %L FEXPR))
                    (PRETTYP (CONS 'DF (CONS %L (CDR %X)))))
                 ((PRETTYP %L)))
              (TERPRI)))))
=  (FEXPR (LAMBDA (%L %X) (MAPC %L '(LAMBDA (%L) (COND ((SETQ %X (GET
= %L EXPR)) (PRETTYP (CONS 'DE (CONS %L (CDR %X))))) ((SETQ %X (GET
= %L FEXPR)) (PRETTYP (CONS 'DF (CONS %L (CDR %X))))) ((PRETTYP %L)))
= (TERPRI)))))

?  (PRETTY PRETTY)

   (DF PRETTY (%L %X)
      (MAPC %L
         '(LAMBDA (%L)
            (COND
               ((SETQ %X (GET %L EXPR))
                  (PRETTYP (CONS 'DE (CONS %L (CDR %X)))))
               ((SETQ %X (GET %L FEXPR))
                  (PRETTYP (CONS 'DF (CONS %L (CDR %X)))))
               ((PRETTYP %L)))
            (TERPRI))))

=  NIL

?  (STOP)
```



## 4.2 LA SORTIE BRAILLE

VLISP-10 possède un mode de sortie en Braille.(*)

Les poinçons sont réalisés par la frappe de caractères spéciaux tels que le point ., la virgule , et le quote ' sur une imprimante qui subit des modifications très légères :

- retrait du ruban encreur ;
- mise d'un "tampon" souple entre les marteaux de l'imprimante et le papier. Ce tampon peut être réalisé au moyen d'une feuille de paravent standard pliée en six.

les meilleurs résultats sont obtenus en utilisant du papier fort.

Il n'existe à l'heure actuelle qu'une fonction d'édition de fichiers : BRAILLEFILE. Pour l'utiliser il faut au préalable charger le fichier

(DSK (BRAILL . VLI)\(630 . 300)).

(BRAILLEFILE *filout filin1 ... filinN*)

édite en braille dans le fichier de sortie *filout*, les différents fichiers d'entrée *filin1 ... filinN*.
BRAILLEFILE ramène NIL.

Des études sur la composition automatique de textes en Braille sont en cours de réalisation à l'Université Paris 8-Vincennes.

(*) *Cette réalisation, unique, n'aurait pas été possible à l'Université de Paris 8-Vincennes, sans la grande patience et les précieuses connaissances sur les problèmes des informaticiens mal-voyants de Messieurs Alain GALLETY et Jean-Marie BUISSON.*





## 4.3 GREDIT

GREDIT est un petit correcteur conversationnel, il est simple et facile à utiliser : son nombre très restreint de commande le rend très commode et très sûr pour modifier des S-expressions. (Le lecteur trouvera un exemple de session complète avec GREDIT en fin de chapitre.)

Les repérages sur les S-expressions arborescentes étant peu pratiques, GREDIT commence par transformer la S-expression à corriger en une suite linéaire d'éléments composée des séparateurs LISP ( ) . et ', et des atomes.

ex : la S-expression

```
        (COND ((NULL L) 'OK)((REP (NEXTL L))))
```

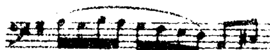
est transformée en une suite d'éléments

```
( COND ( ( NULL L ) ' OK ) ( ( REP ( NEXTL L ) ) ) )
1  2   3 4  5     6 7 8 9 10 . . . .  .  . . . . .
```

Un pointeur courant de position (PCP) contient à tout moment le numéro de l'élément sur lequel l'utilisateur se trouve. Le repérage sur la suite d'éléments se fait par :

- comptage d'éléments à partir du PCP ;
- recherche d'une sous-suite d'éléments, toujours à partir du PCP.

Après avoir exécuté les corrections que l'utilisateur demande au moyen de commandes, GREDIT va transformer la liste d'éléments en une nouvelle S-expression qui, si la transformation réussit, va remplacer la S-expression que l'on voulait corriger. En cas d'échec de cette transformation le contrôle reste à GREDIT.



4.3.1 Les commandes GREDIT

Lorsque GREDIT attend une commande, il imprime une astérisque * sur la TTY, à la place du "prompt-character" standard ?.

Les commandes sont écrites en format libre et doivent tenir sur *une* ligne. Plusieurs commandes peuvent être placées sur une même ligne, mais séparées par des points-et-virgules ;. Les éléments séparateurs (( ) . " ' # ; "RETURN") peuvent être encadrés d'un nombre quelconque d'espaces. Une commande est une suite d'éléments dont le premier est le nom de la commande : seul le premier caractère de ce nom est significatif. Si une commande n'est pas reconnue par GREDIT le message suivant apparaît :

** GREDIT ERROR : INVALID COMMAND = la commande inconnue

Certaines commandes demandent un *repère* sur la chaîne d'éléments qui leur permet de se localiser par rapport au PCP.

| | repères | localisation par rapport au PCP de : |
|---|---|---|
| commande | - rien - | - 1 élément |
| "" | # | - beaucoup d'éléments (jusqu'à la fin de la liste) |
| "" | # *nb* | - *nb* éléments. Si *nb* n'est pas un nombre le message suivant apparaît :<br>** GREDIT ERROR : INVALID COMMAND = # *nb*<br>et la commande n'est pas exécutée |
| "" | *el1* ... *elN* | - jusqu'à la rencontre de la sous-suite *el1* ... *elN*. Si une telle sous-suite n'existe pas le message suivant apparaît :<br>** GREDIT ERROR : SEARCH FAILED = *el1* ... *elN*<br>et la commande n'est pas exécutée. |



4.3.1.1 <u>Commande d'impression P (PRINT)</u>

* P *repère* — imprime les éléments du PCP jusqu'au *repère*. Cette commande ne modifie pas le PCP. Cette impression est précédée de points de suspension si le PCP n'est pas au début de la liste et suivie de points de suspension si tous les éléments de la liste ne sont pas imprimés.

4.3.1.2 <u>Commandes de positionnement > <</u>

* > *repère* — avance dans la suite d'éléments jusqu'au *repère*.

* < *repère* — recule dans la suite d'éléments jusqu'au *repère*.

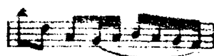
Ces deux commandes modifient le PCP. Après leur exécution GREDIT imprime les 10 premiers éléments à partir du nouveau PCP.

4.3.1.3 <u>Commandes de modification I (INSERT), D (DELETE)</u>

* I *el1 ... elN* — insert à partir du PCP les éléments *el1 ... elN*.

* D *repère* — enlève tous les éléments du PCP jusqu'au *repère*.

Ces deux commandes ne modifient pas le PCP et, comme pour les commandes de positionnement, GREDIT imprime les 10 premiers nouveaux éléments à partir du PCP.



4.3.1.4 Commandes de fin de correction E (END), X (EXIT), S (START)

:: E — indique la fin des corrections. La suite d'éléments est transformée en une S-expression. Si cette transformation échoue le message d'erreur suivant apparaît :

:: GREDIT ERROR : INTERN FAILED

et le contrôle reste à GREDIT ; sinon la nouvelle S-expression remplace l'ancienne.

:: X — sortie extraordinaire de GREDIT. La S-expression corrigée n'est pas transformée, l'ancienne S-expression n'est pas modifiée.

:: S — recommence les corrections sur la S-expression de départ et annule ainsi toutes les précédentes corrections.

4.3.2 Les fonctions GREDIT

(GREDITF *a1* ... *aN*)

appelle GREDIT sur les définitions de fonction des atomes *a1* ... *aN*.

(GREDITV *a1* ... *aN*)

appelle GREDIT sur les valeurs des différents atomes *a1* ... *aN*.



### 4.3.3 Exemple d'utilisation de GREDIT

```
?
?
? (DE FACT (N)
?   (COND ((ZEROP N) N)
?         ((TIMES N (FACT SUB' N))))))
=  FACT
?
? (FACT 3)

  ** UNDEFINED VARIABLE :  SUB

?
? (PRETTY FACT)

    (DE FACT (N)
       (COND
          ((ZEROP N) N)
          ((TIMES N (FACT SUB 'N))))))

=  NIL
?
? (GREDITF FACT)

        ( LAMBDA ( N ) ( COND ( ( ZEROP...
* P#
        ( LAMBDA ( N ) ( COND ( ( ZEROP N ) N ) ( ( TIMES N ( FACT SUB '
  N ) ) ) ) )
* > N)(($D$I 1
       ... N ) ( ( TIMES N ( FACT SUB '...
       ... ) ( ( TIMES N ( FACT SUB ' N...
       ... 1 ) ( ( TIMES N ( FACT SUB '...
* > SUB 'N$D#3$I (SUB1 N)
       ... SUB ' N ) ) ) ) )
       ... ) ) ) ) )
       ... ( SUB1 N ) ) ) ) ) )
* <#$P#
        ( LAMBDA ( N ) ( COND ( ( ZEROP...
        ( LAMBDA ( N ) ( COND ( ( ZEROP N ) 1 ) ( ( TIMES N ( FACT (
  SUB1 N ) ) ) ) ) )
* E

=  NIL
?
? (FACT 3)
=  6
?
? (STOP)
```

## 4.4 PHENARETE (*)

PHENARETE est un programme qui tente de comprendre et d'améliorer des programmes écrits en VLISP. Actuellement, PHENARETE ne travaille que sur un sous-ensemble de VLISP particulièrement adapté à exprimer les connaissances d'un programmeur débutant. PHENARETE ne corrige pas seulement les fautes d'orthographe ou de parenthésage, il essaie en plus de comprendre le programme et fait des propositions afin d'améliorer le programme.

Pour utiliser les fonctions de PHENARETE, il faut au préalable charger le fichier (PHENAR . VLI).

### 4.4.1 Les fonctions de PHENARETE

(PHENARETE $a_1 \ldots a_n$)

commente et améliore (si possible) les fonctions utilisateur $a_1 \ldots a_n$.

(PHENARETES *s*)

commente et améliore la S-expression *s*. Ramène la nouvelle expression corrigée en valeur.

(PHENARETEFILE *filout filin1 ... filinN*)

créé un nouveau fichier *filout* contenant toutes les S-expressions des fichiers *filin1 ... filinN*, traitées par PHENARETE

(*) *PHENARETE a été conçu et réalisé par H. WERTZ : "Sur la Compréhension des programmes LISP améliorables".*

### 4.4.2 Exemples commentés d'utilisation du système

Nous exposons ici quelques exemples illustrant les capacités de compréhension et d'amélioration du système actuel.

#### 4.4.2.1 Fautes d'orthographe

Supposons que l'on ait donné au système la définition de la fonction suivante :

```
(DE FACT (N)
   (COND
     ((ZEROPP N) 1)
     (T (TIMS N (FACCT (UB1 N)) )) ))
```

C'est une définition récursive d'une fonction qui calcule la factorielle d'un nombre N positif.

Cette définition contient plusieurs erreurs d'orthographe :

- ZEROP est écrit ZEROPP .
- TIMES est écrit TIMS
- FACT est à l'appel récursif écrit FACCT et
- SUB1 se présente comme UB1

| Le système répond | explication |
|---|---|
| ERREUR NOM : ?ZEROPP → ZEROP | ne connaissant pas de fonction ZEROPP, φαιναρετη suppose que l'utilisateur a voulu écrire ZEROP |
| ERREUR NOM : ?TIMS → TIMES | elle ne connait pas TIMS, donc elle le remplace par TIMES |
| ERREUR NOM : ?FACCT → FACT | elle remplace FACCT par FACT: FACT est en effet l'unique fonction dont elle a connaissance, et FACCT lui ressemble |
| ERREUR NOM : ?UB1 → SUB1 | même chose |
| PROPOSITION :<br>(DE FACT (N)<br>(COND<br>((ZEROP N) 1)<br>(T (TIMES N(FACT (SUB1 N)))))) | φαιναρετη propose alors son amélioration du programme, ici purement orthographique |





### 4.4.2.2 Fautes de parenthésage

Donnons à φαιναρετη un autre programme :

```
(DE (FACT (N
     (COND
        (ZERO (N (1
        (T (TIMES (NN (FACT (SUB1 (N))) ))) ))) ))) )
```

Il s'agit de la même fonction récursive, avec un parenthésage vraiment fantaisiste ...

| φαιναρετη propose | actions entreprises |
|---|---|
| (DE FACT (N) | elle a reconnu que FACT est la fonction et N la variable |
| (COND<br>((ZEROP N) 1) | reconnaissance de la première clause du COND et remplacement de ZERO par ZEROP |
| (T (TIMES N (FACT (SUB1 N))<br>)) )) | reconnaissance de la deuxième clause du COND. Ne connaissant pas de variable NN elle l'a remplacée par N |

φαιναρετη a rectifié correctement cette syntaxe inhabituelle, et livré une définition de la fonction FACT tout à fait acceptable.

### 4.4.2.3 Vérification sémantique des fonctions de contrôle

Exemple sur COND

```
(DE (FACT) (N
     COND
       (T TIMES N FACT SUB1 N))
       (ZEROP N 1))
```

Ce n'est guère mieux :

- le parenthésage laisse tout autant à désirer ...
- les clauses du COND sont inversées.

| φαιναρετη propose | actions entreprises |
|---|---|
| (DE FACT (N)<br>(COND } | correction des erreurs de parenthésage |
| ((ZEROP N) 1)<br>(T (TIMES N (FACT (SUB1 N))<br>)) )) | remise en place des clauses du COND et correction des erreurs de parenthésage |



### 4.4.2.4 Contrôle des boucles

Donnons au système une quatrième version de FACT :

```
(DE FACT N RES
    (SETQ RES 1)
    (WHILE ((GT N 1))
          (SETQ RES (TIMES N RES)))
RES)
```

Outre les erreurs de parenthésage, cette définition a le grave inconvénient de ne pas faire varier N à l'intérieur de la boucle WHILE , cette boucle est donc infinie.

| φαιναρετη propose | actions entreprises |
|---|---|
| (DE FACT (N RES)<br>(SETQ RES 1) }<br>(WHILE (GT N 1) | corrections des erreurs de parenthésage |
| (SETQ RES (TIMES N RES))<br>(SETQ N (SUB1 N)) )<br>RES) | introduction d'une modification de la variable de contrôle de la boucle : le programme ne "boucle" plus |

### 4.4.2.5. Amélioration sémantique d'une fonction-utilisateur

Un dernier exemple (le commentaire se trouve dans le texte)

```
(DE INVERSER (L1 L2)
    (COND
       ((NULL L2) L1)
       (T
         (INVERSER
               (CDR L1)
               (CONS (CAR L1) L2))) ))
```

Le lecteur appréciera la subtile réponse de φαιναρετη :

```
(DE INVERSER (L1 L2)
    (COND
       ((NULL L1) L2)
       (T
         (INVERSER
               (CDR L1)
               (CONS (CAR L1) L2) )) ))
```

## REFERENCES BIBLIOGRAPHIQUES


FENTCHEL R. : *On implementation of Label Variables*, Comm. ACM 14,5 (May 1971), pp. 349-350.

GREUSSAY P. : *LISP T1600 : manuel de référence*, Département d'Informatique, Université de Paris 8, Février 1975.

GREUSSAY P. : *VLISP : structure et extension d'un système LISP pour mini-ordinateurs*, RT 16-76, Département d'Informatique, Université de Paris 8, Janvier 1976.

GREUSSAY P. : *Descriptions compactes d'interprètes implémentables*, 2ème Colloque International sur la Programmation, Avril 1976, ed. B. ROBINET, Avril 1976, Paris, pp. 281-297.

HEWITT C. et al. : *Behavioral Semantics of non-recursive Control Structures*, Proc. Colloque sur la programmation, ed. B. ROBINET, Avril 1974, Paris, pp. 385-407.

HEWITT C. : *Stereotype as an Actor Approach towards Solving the Problem of Procedural Attachment in FRAME Theories*, Proc 4th IJCAI, 1975, TBILISI, pp. 108-117.

HUET G.P. : *A Unification Algorithm for Typed Lambda-Calculus*, Theoritical Computer Sciences 1, 1975, pp. 27-57.

Mc CARTHY J. : *Recursive Function of Symbolic Expressions and theirs Computations by Machine*, Comm. ACM 3,3, (March 1960), pp. 184 195.

Mc CARTHY J. et al. : *LISP 1.5 Programmer's Manual*, the MIT Press, Cambridge, Mass., 1964.

Mc DERMOTT D.V., SUSSMAN G.J. : *The CONNIVER Reference Manual*, MIT AI Laboratory, Mémo n° 259a, January 1974.

MOON D.A. : *MACLISP Reference Manual*, Cambridge, MIT Project MAC, December 1975.

NEWEY M.C. : *Formal Semantics of LISP with Applications to Program Correctness*, Artificial Intelligence Laboratory AIM-257, Stanford University, January 1975.

QUAM L.H., DIFFIE W. : *Stanford LISP 1.6 Manual*, Stanford Artificial Intelligence Laboratory Operating Note 28.6, Stanford University, 1972.

REYNOLDS J.C. : *Definitionnal Interpreters for High-Order Programming Languages*, Proc. of ACM National Meeting, August 1972, Boston.

TEITELMAN W. : *INTERLISP Reference Manual*, Xerox Palo Alto Research Center, (October 1974).



WHITE J.L. : *An Interim LISP User's Guide*, MIT AI Laboratory Memo n°190, March 1970.

WERTZ H. : *Sur la compréhension des programmes LISP améliorables*, Département d'Informatique, Université de Paris 8, Mars 1976, RT 18-76.

INDEX DES FONCTIONS STANDARD

INDEX DES FONCTIONS STANDARD

INDEX DES FONCTIONS STANDARD

| | FONCTION | TYPE | ARG | SECT | PAGE |
|---|---|---|---|---|---|
| 90 | LMARGIN | STATUS | 1 | 2 . 15 | 62 |
| 91 | LOC | SUBR | 1 | 2 . 16 | 73 |
| 92 | LT | SUBR | N | 2 . 12 | 47 |
| 93 | LZP | SUBR | 1 | 2 . 12 | 47 |
| 94 | MACHAR | STATUS | 2 | 2 . 15 | 66 |
| 95 | MAKLIST | SUBR | 1 | 3 . 1 | 77 |
| 96 | MAP | SUBR | 2 | 2 . 4 | 31 |
| 97 | MAPC | SUBR | 2 | 2 . 4 | 31 |
| 98 | MAPCAR | SUBR | 2 | 2 . 4 | 31 |
| 99 | MAPCT | SUBR | 2 | 2 . 4 | 31 |
| 100 | MAPLIST | SUBR | 2 | 2 . 4 | 31 |
| 101 | MAPS | SUBR | 2 | 2 . 4 | 31 |
| 102 | MAPST | SUBR | 2 | 2 . 4 | 31 |
| 103 | MAPSUB | SUBR | 2 | 2 . 4 | 31 |
| 104 | MAPT | SUBR | 2 | 2 . 4 | 31 |
| 105 | MAX | SUBR | N | 2 . 12 | 49 |
| 106 | MEMBER | SUBR | 2 | 2 . 5 | 33 |
| 107 | MEMQ | SUBR | 2 | 2 . 5 | 32 |
| 108 | MIN | SUBR | N | 2 . 12 | 49 |
| 109 | MINUS | SUBR | 1 | 2 . 12 | 48 |
| 110 | NCONC | SUBR | 2 | 2 . 7 | 39 |
| 111 | NCONC1 | SUBR | N | 2 . 7 | 39 |
| 112 | NEGBIT | STATUS | N | 2 . 15 | 61 |
| 113 | NEQ | SUBR | 2 | 2 . 2 | 21 |
| 114 | NEQP | SUBR | 2 | 2 . 2 | 20 |
| 115 | NEQUAL | SUBR | 2 | 2 . 2 | 21 |
| 116 | NEROP | SUBR | 1 | 2 . 12 | 47 |
| 117 | NEWL | FSUBR | 2 | 2 . 7 | 38 |
| 118 | NEXTL | FSUBR | 1 | 2 . 7 | 38 |
| 119 | NOT | SUBR | 1 | 2 . 2 | 20 |
| 120 | NTH | SUBR | 2 | 2 . 5 | 33 |
| 121 | NULL | SUBR | 1 | 2 . 2 | 20 |
| 122 | NULLSTRP | SUBR | 1 | 3 . 1 | 78 |
| 123 | NUMBP | SUBR | 1 | 2 . 12 | 47 |
| 124 | OBASE | STATUS | 1 | 2 . 15 | 62 |
| 125 | OBLIST | SUBR | 1 | 2 . 6 | 36 |
| 126 | ODDP | SUBR | 1 | 2 . 12 | 47 |
| 127 | OR | FSUBR | N | 2 . 3 | 22 |
| 128 | ORF | SUBR | N | 2 . 4 | 30 |
| 129 | OUTPUT | SUBR | 1 | 2 . 14 | 53 |
| 130 | PAGE | SUBR | 0 | 2 . 14 | 56 |
| 131 | PAIRLIS | SUBR | 3 | 2 . 8 | 42 |
| 132 | PBUFOUT | STATUS | 1 | 2 . 15 | 63 |
| 133 | PEEKCH | SUBR | 0 | 2 . 14 | 55 |
| 134 | PHENARETE | PHENAR | N | 4 . 4 | 91 |

INDEX DES FONCTIONS STANDARD

# INDEX DES FONCTIONS STANDARD